User's Manual

SL1000 Control API

YOKOGAWA
Yokogawa Electric Corporation

IM 720320-01E
1st Edition

Thank you for purchasing the SL1000 Control API.
The SL1000 Control API (SxAPI) is an API (application programming interface) library for creating measurement control application programs for the SL1000 High-Speed Data Acquisition Unit.
SxAPI is provided through a Microsoft Windows dynamic link library (DLL). You can use it in WIN32 development environments such as Microsoft Visual C++ and Microsoft Visual Basic, or in Microsoft .NET Framework development environments such as Microsoft Visual C# and Microsoft Visual Basic 2005.
This user's manual explains the specifications of the SL1000 Control API interface.
Keep this manual in a safe place for quick reference in the event a question arises. The following manuals, including this one, are provided as manuals for the SL1000. Please read all of them.

Manual Title	Manual No.	Description
SL1000 High-Speed Data Acquisition Unit User's Manual	IM 720120-01E	Explains how to install the SL1000 and its input modules, and explains features related to the hardware, such as the display, and how to operate them.
SL1000 Acquisition Software User's Manual	IM 720120-61E	Explains all functions and procedures of the Acquisition Software used to configure and control the SL1000.
SL1000 Input Module User's Manual	IM 720120-51E	Explains the specifications of the input modules that can be installed in the SL1000.
701992/701994 Xviewer User's Manual	IM 701992-01E	Explains all functions and procedures of the Xviewer software used to display the measured data as waveforms on a PC. This manual is not included with the /XV0 option.
SL1000 Control API User's Manual	IM 720320-01E	This manual. It explains the functions for controlling the SL1000 (the SL1000 control API).

Notes

- The contents of this manual apply to the SL1000 Control API Ver. 1.01. If you use another version of the SL1000 Control API, its contents may be different than those of the Control API described in this manual.
- The contents of this manual are subject to change without prior notice as a result of continuing improvements to the instrument's performance and functionality. The figures given in this manual may differ from those that actually appear on your screen.
- Every effort has been made in the preparation of this manual to ensure the accuracy of its contents. However, should you have any questions or find any errors, please contact your nearest YOKOGAWA dealer.
- Copying or reproducing all or any part of the contents of this manual without the permission of YOKOGAWA is strictly prohibited.

1st Edition: May 2008(YK)
All Rights Reserved, Copyright © 2007 Yokogawa Electric Corporation

Trademark Acknowledgements

- Microsoft, Windows, Active X, Visual Basic, ,Visual C#, and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated.
- In this manual, the TM and ® symbols do not accompany their respective registered trademark or trademark names.
- Other company and product names are registered trademarks or trademarks of their respective holders.

Revisions

- 1st Edition: May 2008

Terms and Conditions of the Software License

Yokogawa Electric Corporation, a Japanese corporation (hereinafter called “Yokogawa”), grants permission to use this Yokogawa Software Program (hereinafter called the “Licensed Software”) to the Licensee on the conditions that the Licensee agrees to the terms and conditions stipulated in Article 1 hereof.
You, as the Licensee (hereinafter called “Licensee”), shall agree to the following terms and conditions for the software license (hereinafter called the “Agreement”) based on the use intended for the Licensed Software.
Please note that Yokogawa grants the Licensee permission to use the Licensed Software under the terms and conditions herein and in no event shall Yokogawa intend to sell or transfer the Licensed Software to the Licensee.
Licensed Software Name: SL1000 Control API
Number of License: 1

Article 1 (Scope Covered by these Terms and Conditions)
1.1 The terms and conditions stipulated herein shall be applied to any Licensee who purchases the Licensed Software on the condition that the Licensee consents to agree to the terms and conditions stipulated herein.
1.2 The “Licensed Software” herein shall mean and include all applicable programs and documentation, without limitation, all proprietary technology, algorithms, and know-how such as a factor, invariant or process contained therein.

Article 2 (Grant of License)
2.1 Yokogawa grants the Licensee, for the purpose of single use, non-exclusive and non-transferable license of the Licensed Software with the license fee separately agreed upon by both parties.
2.2 The Licensee is, unless otherwise agreed in writing by Yokogawa, not entitled to copy, change, sell, distribute, transfer, or sublicense the Licensed Software.
2.3 The Licensed Software shall not be copied in whole or in part except for keeping one (1) copy for back-up purposes. The Licensee shall secure or supervise the copy of the Licensed Software by the Licensee itself with great, strict, and due care.
2.4 In no event shall the Licensee dump, reverse assemble, reverse compile, or reverse engineer the Licensed Software so that the Licensee may translate the Licensed Software into other programs or change it into a man-readable form from the source code of the Licensed Software. Unless otherwise separately agreed by Yokogawa, Yokogawa shall not provide the Licensee the source code for the Licensed Software.
2.5 The Licensed Software and its related documentation shall be the proprietary property or trade secret of Yokogawa or a third party which grants Yokogawa the rights. In no event shall the Licensee be transferred, leased, sublicensed, or assigned any rights relating to the Licensed Software.
2.6 Yokogawa may use or add copy protection in or onto the Licensed Software. In no event shall the Licensee remove or attempt to remove such copy protection.
2.7 The Licensed Software may include a software program licensed for re-use by a third party (hereinafter called “Third Party Software”, which may include any software program from affiliates of Yokogawa made or coded by themselves.) In the case that Yokogawa is granted permission to sublicense to third parties by any licensors (sub-licensor) of the Third Party Software pursuant to different terms and conditions than those stipulated in this Agreement, the Licensee shall observe such terms and conditions of which Yokogawa notifies the Licensee in writing separately.
2.8 In no event shall the Licensee modify, remove or delete a copyright notice of Yokogawa and its licenser contained in the Licensed Software, including any copy thereof.

Article 3 (Restriction of Specific Use)
3.1 The Licensed Software shall not be intended specifically to be designed, developed, constructed, manufactured, distributed or maintained for the purpose of the following events:
  a) Operation of any aviation, vessel, or support of those operations from the ground;,
  b) Operation of nuclear products and/or facilities;,
  c) Operation of nuclear weapons and/or chemical weapons and/or biological weapons; or
  d) Operation of medical instrumentation directly utilized for humankind or the human body.
3.2 Even if the Licensee uses the Licensed Software for the purposes in the preceding Paragraph 3.1, Yokogawa has no liability to or responsibility for any demand or damage arising out of the use or operations of the Licensed Software, and the Licensee agrees, on its own responsibility, to solve and settle the claims and damages and to defend, indemnify or hold Yokogawa totally harmless, from or against any liabilities, losses, damages and expenses (including fees for recalling the Products and reasonable attorney’s fees and court costs), or claims arising out of and related to the above-said claims and damages.

Article 4 (Warranty)
4.1 The Licensee shall agree that the Licensed Software shall be provided to the Licensee on an “as is” basis when delivered. If defect(s), such as damage to the medium of the Licensed Software, attributable to Yokogawa is found, Yokogawa agrees to replace, free of charge, any Licensed Software on condition that the defective Licensed Software shall be returned to Yokogawa’s specified authorized service facility within seven (7) days after opening the Package at the Licensee’s expense. As the Licensed Software is provided to the Licensee on an “as is” basis when delivered, in no event shall Yokogawa warrant that any information on or in the Licensed Software, including without limitation, data on computer programs and program listings, be completely accurate, correct, reliable, or the most updated.
4.2 Notwithstanding the preceding Paragraph 4.1, when third party software is included in the Licensed Software, the warranty period and terms and conditions that apply shall be those established by the provider of the third party software.
4.3 When Yokogawa decides in its own judgement that it is necessary, Yokogawa may from time to time provide the Licensee with Revision upgrades and Version upgrades separately specified by Yokogawa (hereinafter called “Updates”).
4.4 Notwithstanding the preceding Paragraph 4.3, in no event shall Yokogawa provide Updates where the Licensee or any third party conducted renovation or improvement of the Licensed Software.
4.5 THE FOREGOING WARRANTIES ARE EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES OF QUALITY AND PERFORMANCE, WRITTEN, ORAL, OR IMPLIED, AND ALL OTHER WARRANTIES INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE ARE HEREBY DISCLAIMED BY YOKOGAWA AND ALL THIRD PARTIES LICENSING THIRD PARTY SOFTWARE TO YOKOGAWA.
4.6 Correction of nonconformity in the manner and for the period of time provided above shall be the Licensee’s sole and exclusive remedy for any failure of Yokogawa to comply with its obligations and shall constitute fulfillment of all liabilities of Yokogawa and any third party licensing the Third Party Software to Yokogawa (including any liability for direct, indirect, special, incidental or consequential damages) whether in warranty, contract, tort (including negligence but excluding willful conduct or gross negligence by Yokogawa) or otherwise with respect to or arising out of the use of the Licensed Software.

Article 5 (Infringement)
5.1 If and when any third party should demand injunction, initiate a law suit, or demand compensation for damages against the Licensee under patent right (including utility model right, design patent, and trade mark), copy right, and any other rights relating to any of the Licensed Software, the Licensee shall notify Yokogawa in writing to that effect without delay.
5.2 In the case of the preceding Paragraph 5.1, the Licensee shall assign to Yokogawa all of the rights to defend the Licensee and to negotiate with the claiming party. Furthermore, the Licensee shall provide Yokogawa with necessary information or any other assistance for Yokogawa’s defense and negotiation. If and when such a claim should be attributable to Yokogawa, subject to the written notice to Yokogawa stated in the preceding Paragraph 5.1, Yokogawa shall defend the Licensee and negotiate with the claiming party at Yokogawa’s cost and expense and be responsible for the final settlement or judgment granted to the claiming party in the preceding Paragraph 5.1.
5.3 When any assertion or allegation of the infringement of the third party’s rights defined in Paragraph 5.1 is made, or when at Yokogawa’s judgment there is possibility of such assertion or allegation, Yokogawa will, at its own discretion, take any of the following countermeasures at Yokogawa’s cost and expense.
  a) To acquire the necessary right from a third party which has lawful ownership of the right so that the Licensee will be able to continue to use the Licensed Software;
  b) To replace the Licensed Software with an alternative one which avoids the infringement; or
  c) To remodel the Licensed Software so that the Licensed Software can avoid the infringement of such third party’s right.
5.4 If and when Yokogawa fails to take either of the countermeasures as set forth in the preceding subparagraphs of Paragraph 5.3, Yokogawa shall indemnify the Licensee only by paying back the price amount of the Licensed Software which Yokogawa has received from the Licensee. THE FOREGOING PARAGRAPHS STATE THE ENTIRE LIABILITY OF YOKOGAWA AND ANY THIRD PARTY LICENSING THIRD PARTY SOFTWARE TO YOKOGAWA WITH RESPECT TO INFRINGEMENT OF THE INTELLECTUAL PROPERTY RIGHTS INCLUDING BUT NOT LIMITED TO, PATENT AND COPYRIGHT.

Article 6 (Liabilities)

6.1 If and when the Licensee should incur any damage relating to or arising out of the Licensed Software or service that Yokogawa has provided to the Licensee under the conditions herein due to a reason attributable to Yokogawa, Yokogawa shall take actions in accordance with this Agreement. However, in no event shall Yokogawa be liable or responsible for any special, incidental, consequential and/or indirect damage, whether in contract, warranty, tort, negligence, strict liability, or otherwise, including, without limitation, loss of operational profit or revenue, loss of use of the Licensed Software, or any associated products or equipment, cost of capital, loss or cost of interruption of the Licensee's business, substitute equipment, facilities or services, downtime costs, delays, and loss of business information, or claims of customers of Licensee or other third parties for such or other damages. Even if Yokogawa is liable or responsible for the damages attributable to Yokogawa and to the extent of this Article 6, Yokogawa's liability for the Licensee's damage shall not exceed the price amount of the Licensed Software or service fee which Yokogawa has received. Please note that Yokogawa shall be released or discharged from part or all of the liability under this Agreement if the Licensee modifies, remodels, combines with other software or products, or causes any deviation from the basic specifications or functional specifications, without Yokogawa's prior written consent.

6.2 All causes of action against Yokogawa arising out of or relating to this Agreement or the performance or breach hereof shall expire unless Yokogawa is notified of the claim within one (1) year of its occurrence.

6.3 In no event, regardless of cause, shall Yokogawa assume responsibility for or be liable for penalties or penalty clauses in any contracts between the Licensee and its customers.

Article 7 (Limit of Export)

Unless otherwise agreed by Yokogawa, the Licensee shall not directly or indirectly export or transfer the Licensed Software to any countries other than those where Yokogawa permits export in advance.

Article 8 (Term)

This Agreement shall become effective on the date when the Licensee receives the Licensed Software and continues in effect unless or until terminated as provided herein, or the Licensee ceases using the Licensed Software by itself or with Yokogawa's thirty (30) days prior written notice to the Licensee.

Article 9 (Injunction for Use)

During the term of this Agreement, Yokogawa may, at its own discretion, demand injunction against the Licensee in case that Yokogawa deems that the Licensed Software is used improperly or under severer environments other than those where Yokogawa has first approved, or any other condition which Yokogawa may not permit.

Article 10 (Termination)

Yokogawa, at its sole discretion, may terminate this Agreement without any notice or reminder to the Licensee if the Licensee violates or fails to perform this Agreement. However, Articles 5, 6, and 11 shall survive even after the termination.

Article 11 (Jurisdiction)

Any dispute, controversies, or differences between the parties hereto as to interpretation or execution of this Agreement shall be resolved amicably through negotiation between the parties upon the basis of mutual trust. Should the parties fail to agree within ninety (90) days after notice from one of the parties to the other, both parties hereby irrevocably submit to the exclusive jurisdiction of the Tokyo District Court (main office) in Japan for settlement of the dispute.

Article 12 (Governing Law)

This Agreement shall be governed by and construed in accordance with the laws of Japan. The Licensee expressly agrees to waive absolutely and irrevocably and to the fullest extent permissible under applicable law any rights against the laws of Japan which it may have pursuant to the Licensee's local law.

Article 13 (Severability)

In the event that any provision hereof is declared or found to be illegal by any court or tribunal of competent jurisdiction, such provision shall be null and void with respect to the jurisdiction of that court or tribunal and all the remaining provisions hereof shall remain in full force and effect.

How to Use This Manual

Structure of the Manual

This manual contains five chapters, an appendix, and an index.

Chapter	Title	Description
1	**General Features**	Explains the general features of SxAPI.
2	**Events**	Explains event conditions and programming.
3	**Function Details**	Explains the details of all of the SxAPI functions.
4	**SL1000 Communication Commands**	Explains the communication commands that the SL1000 supports.
5	**Error Codes**	Explains SxAPI error codes and SL1000 error codes.
	Appendix	Contains sample programs.
	Index	

Contents

Terms and Conditions of the Software License iii
How to Use This Manual v

Chapter 1 General Features

1.1 File Structure and Operating Environment 1-1
1.2 Programming with SxAPI 1-2
1.3 Programming Flowchart 1-4
1.4 .NET Control Library 1-5

Chapter 2 Events

2.1 Event Conditions 2-1
2.2 Programming Methods 2-2

Chapter 3 Function Details

3.1 Functions 3-1
3.2 Initialization and Ending 3-6
3.3 Retrieving Device Information 3-8
3.4 Opening and Closing Handles 3-10
3.5 Handle Acquisition 3-12
3.6 Value Retrieval 3-15
3.7 Measuring Group Settings 3-17
3.8 Communication Command Controls 3-18
3.9 Event Controls 3-23
3.10 Measurement Condition Settings and Queries 3-25
3.11 Auto Recording Condition Settings and Queries 3-33
3.12 Measurement Controls 3-43
3.13 Auto Recording Controls 3-45
3.14 Acquisition and Deletion of Measured Data 3-46
3.15 Setup Data Access 3-53
3.16 System-Related Functions 3-54
3.17 Internal Media Operations 3-60
3.18 Debugging 3-65
3.19 Definitions 3-68

Chapter 4 SL1000 Communication Commands

4.1 Using Communication Commands 4-1
4.2 Commands 4-2
4.3 ALARm Group 4-6
4.4 CHANnel Group 4-9
4.5 GONogo Group 4-20
4.6 MEASure Group 4-23
4.7 TRIGger Group 4-25

Chapter 5 Error Codes

5.1 Library Errors 5-1
5.2 Unit Errors 5-2

Appendix

Appendix 1 Sample Programs App-1

Index

1.1 File Structure and Operating Environment

Folder and File Structure

Folder	File Name	Redistribution	Description
DLL	SxAPI.dll tmctl.dll YKMUSB.dll	Yes	The dynamic link libraries necessary to execute a program. Put them in the same folder as the executable program (.exe) files.
VC++	SxAPI.h SxAPI.lib	No	The header and library files that are necessary when programming in C or C++.
VB6	SxAPI.bas	No	The standard module necessary for programming in VB6.
NET	SxAPI.NET.dll	Yes	The .NET control library necessary for programming or executing VB.NET or VC# programs. Put it in the same folder as the executable program (.exe) files.
ActiveX	SxEvent.ocx	Yes	The ActiveX control used to process SxAPI asynchronous messages in VB6. Register it beforehand with RegSvr32.exe.
USBDriver	ykmusb.Inf YKMUSB.sys	Yes	The necessary device drivers for USB connection.
Samples	VC++ VB6 VC# VB.NET	No	Sample programs in various programming languages. There are sample programs for VC++ 6.0, VB 6.0, VC# 2005, and VB.NET 2005.
Documents	IM720320-01E.pdf	No	The user's manual.

Operating Environment

OS

Microsoft Windows 2000 Pro, Windows XP, or Windows Vista

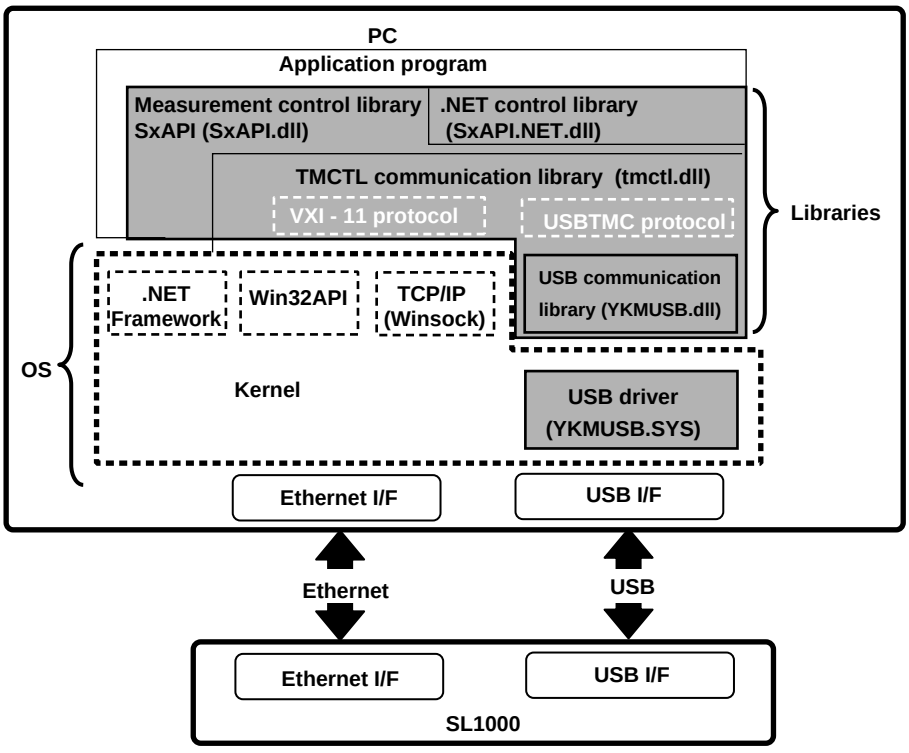
You must install Microsoft .NET Framework version 2.0 or later to use the .NET control library in Windows 2000 Pro or Windows XP.

Development Environment

Microsoft Visual C++ 6.0 or later, Microsoft Visual Basic 6.0,
Microsoft Visual Basic 2005 or later, or Microsoft Visual C# 2005 or later

Position

The diagram below shows the position of the API in relation to an application program. The grey area contains software that the API provides.



1.2 Programming with SxAPI

Handles

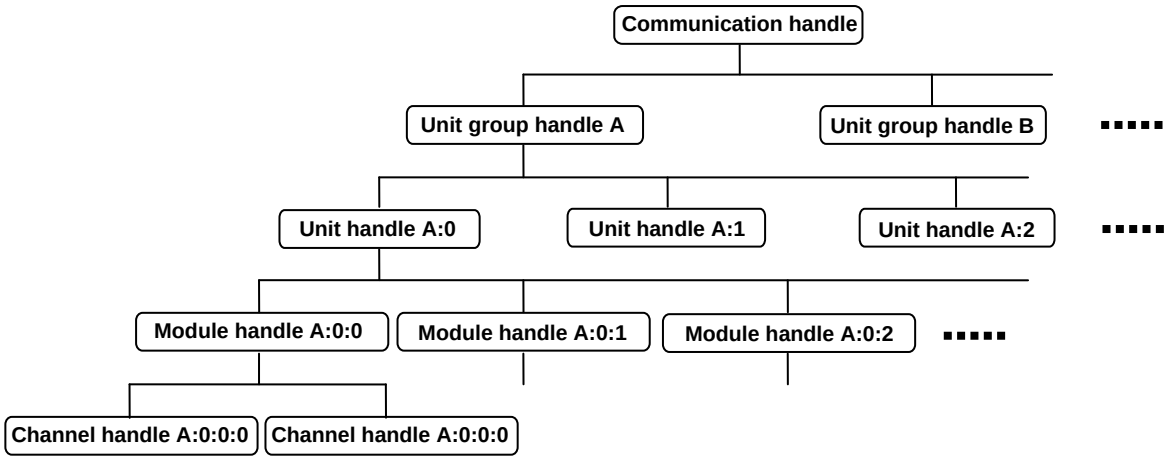
SxAPI is a handle-based API.
In general, a file is accessed when its file name is specified and the file is opened. Then, a file handle is obtained that is used for reading and writing data. Communication types and group IDs are specified and opened (connected to) in the same way. The communication handles and unit group handles that are obtained are used to acquire information, make measurement settings, and control execution.
To make unit, module, and channel access easy, SxAPI handles have the following features.

1. A unit group composed of multiple units can be specified with a single handle (a unit group handle). Because of this, through the execution of a single function, the functions in the same unit group can be accessed.
2. Unit handles, module handles, and channel handles can be acquired from unit group handles. Through the use of different handles for different settings, the appropraite items can be accessed efficiently.
3. Properties such as communication types, group IDs, slot numbers, and channel numbers are managed and processed logically, so programs are not effected by the network connection location or the module installation slot.
   This makes it possible to develop very stable programs.

The kinds of handles are listed below.

Handle Type and Notation	Description
Communication handle SX_HNDL_COMM	This handle is acquired by using SxInit() to establish a connection. It handles communication. Be sure to disconnect by using SxExit() before closing a program.
Unit group handle SX_HNDL_GROUP	This handle is acquired by using SxOpenGroup() to open a unit group. It handles the open unit group. Be sure to close the unit group by using SxCloseGroup() before disconnecting.
Unit handle SX_HNDL_UNIT	This handle is acquired by using SxUnitHndl() or SxMyUnitHndl(). It handles a single unit. There is no close operation.
Module handle SX_HNDL_MOD	This handle is acquired by using SxModHndl() or SxMyModHndl(). It handles a single module. There is no close operation.
Channel handle SX_HNDL_CH	This handle is acquired by using SxChHndl(). It handles a single channel. There is no close operation.
Measuring group handle SX_HNDL_MEASGRP	This handle is acquired by using SxMeasgrpHndl() or SxMyMeasgrpHndl(). It handles a single measuring group. There is no close operation.

The relationships between handles are illustrated in a tree structure on the next page.



Settings and Queries

In addition to handle control, the SxAPI offers functions for setting time-axis-related measurement conditions such as the measuring mode, sampling rate, and recording time, for querying, and for acquiring and saving measured data.

There are settings that relate to the vertical axis, such as the voltage range, coupling, and trigger level, that are available depending on the module. When using these settings and settings that are not provided through specialized functions, such as alarm, waveform parameter computation, and GO/NO-GO judgment, send and recieve unit communication commands through API commands such as SxSetControl() and SxGetControl().

For unit communication command specifications, see chapter 4.

In this API, the reply to unit querry commands is automatically set to “data only” (in other words, “no header”) when a unit is opened.

Asynchronous Messages (Events)

To increase the efficiency of application program execution, SxAPI can indicate trigger detection and measurement completion with window messages. These indications are referred to as events.

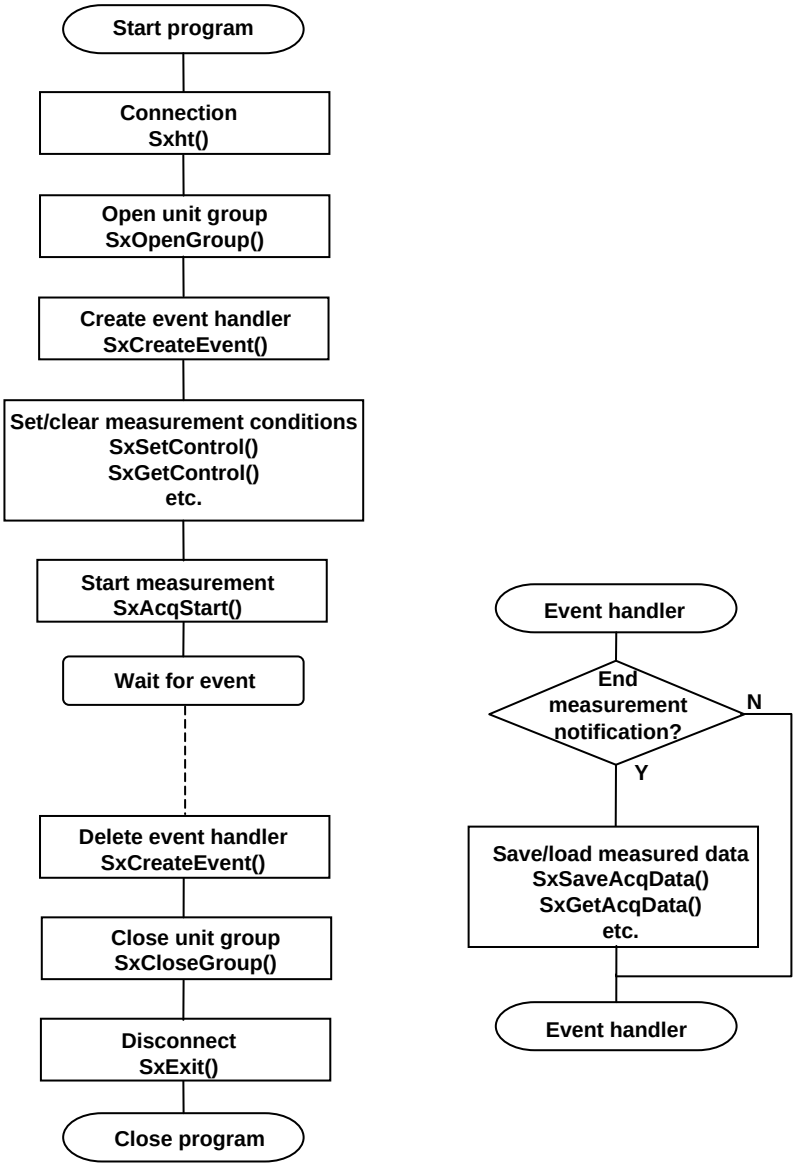
Events are produced using unit SRQ interrupts.

This allows you to write programs that run based on received events. Also, to make this kind of message processing easy, we have also prepared an OCX that can be used in Visual Basic 6.0. In the .NET control library, SRQ notification takes place through the event method.

For details, see chapter 2.

1.3 Programming Flowchart

The following diagram is a basic flowchart of an application program that uses SxAPI.



1.4 .NET Control Library

For programming in a .NET environment, the SxAPI.NET.dll control library is used.
The control library specifications are as follows.

Namespace

namespace SxAPI

Classes

The following two classes are available:

class SxAPI	Provides all functions as methods.
class SxEventArgs	Is a delegate class that provides event parameters to the Event method.

Methods

All functions are provided as SxAPI class methods.
In Chapter 3, “Function Details,” .NET method interfaces are printed in italics.
As a rule, the method that corresponds to a VC++/VB6 function name is simply the function name with the preceding “Sx” removed. Explanations in this manual use VC++/VB6 function names. When programming in a .NET environment, remove the “Sx” from function names.

Asynchronous Messages Notification

Asynchronous messaging is accomplished through the “Event” method.
For details, see chapter 2.

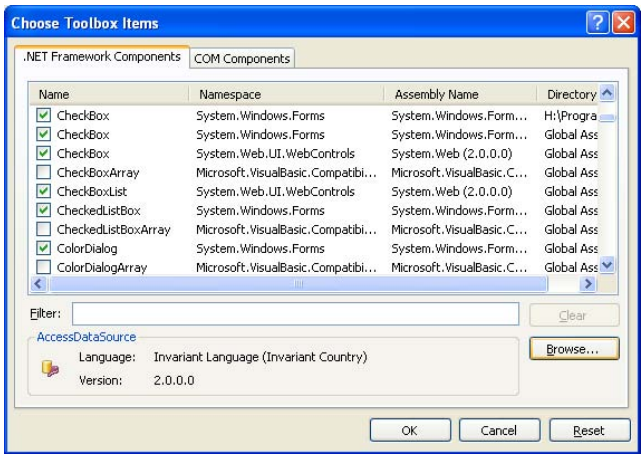
Structures and Constants

Structure and constant definition is fundamentally the same as in VC++/VB6, but constants are defined using enum.
Since .NET structures and constants can be easily inferred from the structures and constants listed in this manual for VC++, they have been left out of the manual.

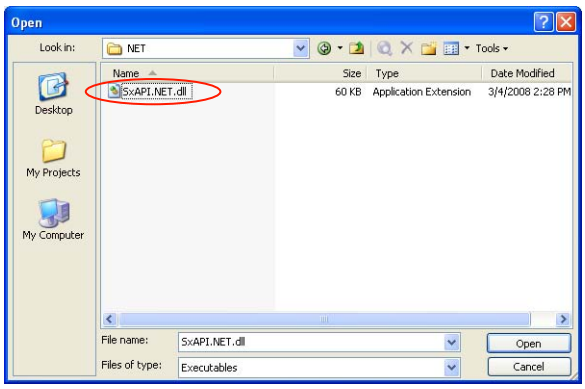
Using the Control Library (Reference)

1. In Visual Studio, click **Tools** then **Choose Toolbox Items** to open the **Choose Toolbox Items** dialog box. Click **Browse** in the **.NET Framework Components** tab.

The **Open** dialog box appears.

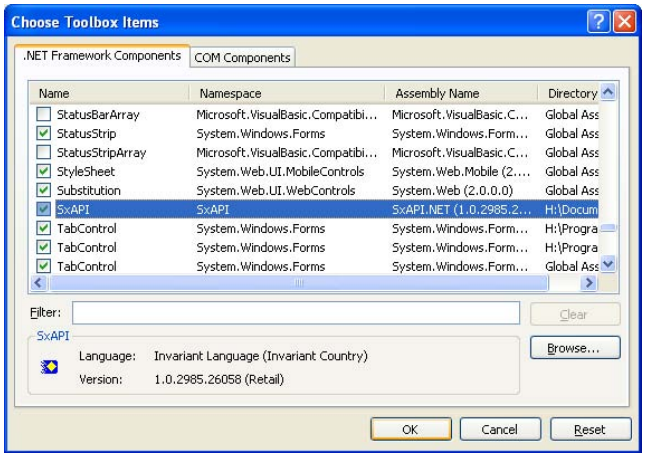


2. Select **SxAPI.NET.dll**, and then click **Open**.

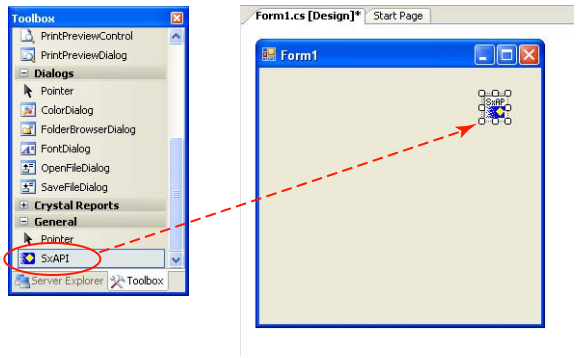


3. Select the **SxAPI** check box, and then click **OK**.

SxAPI appears in the **Toolbox**.



4. Drag the **SxAPI** icon from the **Toolbox** to the desired form.



2.1 Event Conditions

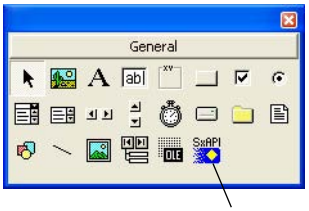
An application program receives an SxAPI event as a 32-bit integer with the conditions that triggered the event assigned to the integer's bits. Application programs may receive multiple event conditions at the same time. If two events with the same condition are raised in brief succession, they may be consolidated into the same event. In other words, the number of times that an application program receives an event may be less than the number of times that the event was actually raised.
You can specify the conditions that you want to be notified of when you generate an event handler using SxCreateEvent().
The table below lists the conditions that can trigger events.

Definition (.NET definitions are written in italics)	Bit Assignment	Condition
SX_EV_ACQ_START *EV.ACQ_START*	0x00010000	Measurement started.
SX_EV_ACQ_STOP *EV.ACQ_STOP*	0x00020000	Measurement stopped.
SX_EV_TRIG_START *EV.TRIG_START*	0x00040000	Trigger detected (in trigger mode).
SX_EV_TRIG_END *EV.TRIG_END*	0x00080000	Triggered measurement stopped (occurs after each acquisition).
SX_EV_ACQ_DATA_READY *EV.ACQ_DATA_READY*	0x08000000	Specified number of data points were acquired in free run mode (occurs after each acquisition).
SX_EV_SAVE_SATRT *EV.SAVE_SATRT*	0x02000000	PC auto-recording operation started.
SX_EV_SAVE_END *EV.SAVE_END*	0x10000000	PC auto-recording operation ended.
SX_EV_CHANNEL_ALARM *EV.CHANNEL_ALARM*	0x20000000	Channel alarm occurred.

2.2 Programming Methods

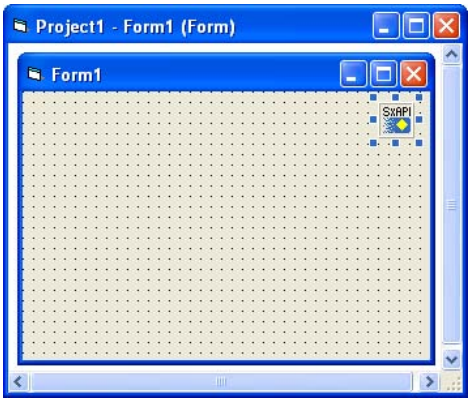
Using SxEvent.ocx in Visual Basic 6.0

Open a project in VB. Click **Project** > **Components**, and then select **SxEvent ActiveX Control Module** from the menu and add it to the project. The icon indicated in the figure below will appear.



SxEvent.ocx

Double-click the SxEvent.ocx icon shown in the figure above and paste it into a form for receiving events. The form should look like the figure below.



Double-click the SxEvent.ocx icon that you pasted into the form to create the event handler.
There is only one event handler type, SRQ1.
The following is an example of a program that saves the most recent measured data to a file if the event condition is SX_EV_TRIG_END (occurs when a triggered measurement ends).

```
Private Sub SxEvent1_SRQ1(ByVal handle As Long, ByVal pattern As Long)

    If pattern And SX_EV_TRIG_END Then ' Save measured data if the event
    ' indicates that a triggered measurement has finished.
       Err = SxSaveAcqData(handle, -1, "C:\Data\filename")
       ' The next trigger is enabled (for when the SL1000 is in Normal
       ' mode)
       Err = SxEnableNextTrig(handle)
    End If

End Sub
```

Using PreTranslateMessage in Visual C++

When the destination window for a message is not specified (NULL is specified) in the first parameter of SxInit(), the message will be sent to the main window.
Main window messages can be processed by overriding CWinApp:: PreTranslateMessage.

```
BOOL WeApiTestApp::PreTranslateMessage(MSG* pMsg)
{
 return CWinApp::PreTranslateMessage(pMsg);
}
```

The sent message ID can be retrieved with RegisterWindowMessage(SX_WM_EVENT). SX_WM_EVENT is defined as follows in SxAPI.h.

```
#define SX_WM_EVENT  "WM_YOKOGAWA_TM_SX_EVENT"
```

Follow the example below to send an SxAPI event to the window that you want it to be handled by.
This example sends an event to CMainFrame.

```
BOOL WeApiTestApp::PreTranslateMessage(MSG* pMsg)
{
  //TODO: Add your specialized code here and/or call the base class.
  UINT msg = pMsg->message;
  if (msg == RegisterWindowMessage(SX_WM_EVENT))
    return m_pMainWnd->SendMessage(msg, pMsg->wParam, pMsg->lParam);
  else
    return CWinApp::PreTranslateMessage(pMsg);
}
```

Define the event handler with the receiving class.

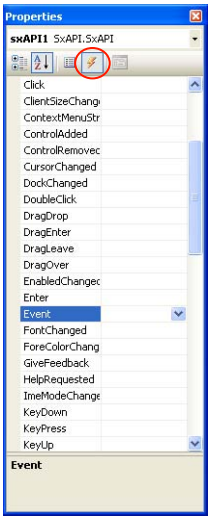
```
const UINT  wm_SxAPI = RegisterWindowMessage(SX_WM_EVENT);
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
  //{{AFX_MSG_MAP(CSampleDlg)
  ON_WM_CREATE()
  ON_WM_CLOSE()
  :
  //}}AFX_MSG_MAP
  ON_REGISTERED_MESSAGE(wm_SxAPI, OnSxEvent)
END_MESSAGE_MAP()
```

Write the event handler.

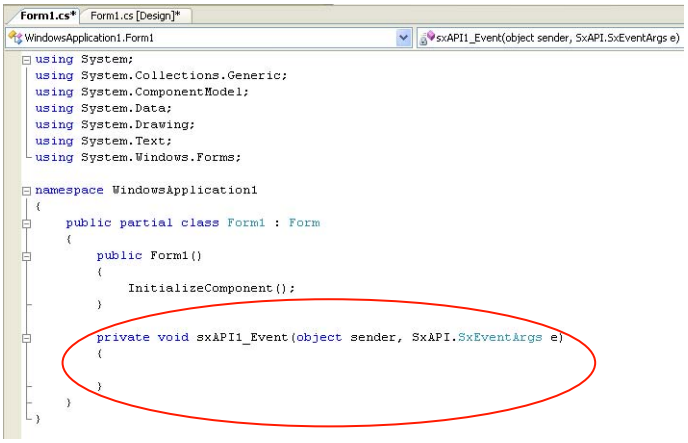
```
LRESULT CMainFrame::OnSxEvent(WPARAM wp, LPARAM lp)
{
  SX_HNDL_UNIT hUnit = wp;// WPARAM is a unit handle.
  ULONG  pattern = lp;    // LPARAM is an event pattern.
  :
  if(pattern & SX_EV_TRIG_END)
  {
    // Handling of the triggered measurement end event.
    :
  }
  :
  return TRUE;
}
```

Using SxAPI.NET in Visual C#

In Visual Studio, drag the SxAPI control onto the form and open **Properties**. Click to display a list of events, and then double-click **Event**.



The following event handling method will appear in the code.



Write the event handling code inside of the above method.
The following is an example of a program that saves the most recent measured data to a file if the event condition is SX_EV_TRIG_END (occurs when a triggered measurement ends).

```
private void sxAPI1_Event(object sender, SxAPI.SxEventArgs e)
{
    int ret;
    if((e.pattern & (uint)EV.TRIG_END) != 0) // When there is a triggered
                                             // measurement end event
    {
        // Save measured data
        ret = sxAPI1.SaveAcqData(e.unitHndl, -1, "C:\\Data\\filename");
        // The next trigger is enabled (for when the SL1000 is in Normal
        // mode)
        ret = sxAPI1.EnableNextTrig(e.unitHndl);
    }
}
```

3.1 Functions

Initialization and Ending

Function	Name	Page
Initialize	SxInit	3-6
Exit	SxExit	3-6
Re-search	SxReSearch	3-7

Retrieving Device Information

Function	Name	Page
Get unit group information	SxGetGroupInfo	3-8
Get number of units	SxGetUnitNum	3-8
Get unit information	SxGetUnitInfo	3-9
	SxGetUnitInfoS	
Get module information	SxGetModInfo;	3-9

Opening and Closing Handles

Function	Name	Page
Open unit group	SxOpenGroup	3-10
	SxOpenGroupEx	
Close unit group	SxCloseGroup	3-10
Open unit	SxOpenUnit	3-11
	SxOpenUnitEx	3-11
Close unit	SxCloseUnit	

Retrieving Communication Handles

Function	Name	Page
Get communication handle	SxMyCommHndl	3-12
Get unit group handle	SxMyGrpHndl	3-12
Get unit handle	SxMyUnitHndl	3-12
Get unit handle (by unit number)	SxUnitHndl	3-12
Get measuring group handle	SxMyMeasgrpHndl	3-13
Get measuring group handle (by measuring group number)	SxMeasgrpHndl	3-13
Get module handle	SxMyModHndl	3-14
	SxModHndl	
Get channel handle	SxChHndl	3-14

Retrieving Values

Function	Name	Page
Get channel number	SxChNo	3-15
Get module number	SxModNo	3-15
Get unit number	SxUnitNo	3-15
Get unit group number	SxGrpNo	3-15
Get measuring group number	SxMeasgrpNo	3-16
Get channel number	SxChNum	3-16
Get number of modules	SxModNum	3-16
Get unit number	SxUnitNum	3-16

Measuring Group Settings

Function	Name	Page
Set up measuring group	SxSetupMeasgrp	3-17

Communication Command Controls

Function	Name	Page
Send command	SxSetControl	3-18
Send binary data	SxSetControlBinary	3-20
Send and receive command	SxGetControl	3-20
Send and receive command (binary data)	SxGetControlBinary	3-21
Get received parameter	SxGetParam	3-21
(Copy to the specified buffer)	SxGetParamStr	
Get received parameter (get location)	SxGetParamPos	3-22

Event Controls

Function	Name	Page
Create and enable event handler	SxCreateEvent	3-23
Delete event handler	SxDeleteEvent	3-23
Enable next trigger and related event notification		
SxEnableNextTrig	3-24	

Measurement Condition Settings and Queries

Function	Name	Page
Switch measurement on/off	SxSetAcqSwitch	3-25
Query measurement on/off	SxGetAcqSwitch	3-25
Switch auto recording on/off	SxSetRecSwitch	3-26
Query auto recording on/off	SxGetRecSwitch	3-26
Set measuring mode	SxSetAcqMode	3-26
Query measuring mode	SxGetAcqMode	3-26
Set sampling clock	SxSetClockSource	3-27
Query sampling clock	SxGetClockSource	3-27
Set sample rate	SxSetSamplingRate	3-27
Query sample rate	SxGetSamplingRate	3-27
Set sample interval	SxSetSamplingInterval	3-28
Query sample interval	SxGetSamplingInterval	3-28
Set sample points	SxSetAcqLength	3-28
Query sample points	SxGetAcqLength	3-28
Set sampling time	SxSetAcqSpan	3-29
Query sampling time	SxGetAcqSpan	3-29
Set trigger mode	SxSetTrigMode	3-29
Query trigger mode	SxGetTrigMode	3-29
Set pre-trigger position	SxSetTrigPos	3-30
Query pre-trigger position	SxGetTrigPos	3-30
Set pre-trigger points	SxSetPretrigLength	3-30
Query pre-trigger points	SxGetPretrigLength	3-30
Set trigger count	SxSetTrigCount	3-31
Query trigger count	SxGetTrigCount	3-31
Set channel label	SxSetChLabel	3-31
Query channel label	SxGetChLabel	3-31
Query acquisition data capacity	SxGetAcqCapacity	3-32

Auto Recording Condition Settings and Queries

Function	Name	Page
Set auto recording destination	SxSetRecDest	3-33
Query auto recording destination	SxGetRecDest	3-33
Set recording start condition	SxSetRecStartCond	3-33
Query recording start condition	SxGetRecStartCond	3-34
Set recording start time	SxSetRecStartTime	3-34
Query recording start time	SxGetRecStartTime	3-34
Set recording stop condition	SxSetRecStopCond	3-35
Query recording stop condition	SxGetRecStopCond	3-35
Set recording end time	SxSetRecStopTime	3-36
Query recording end time	SxGetRecStopTime	3-36
Set recording time	SxSetRecSpan	3-36
Query recording time	SxGetRecSpan	3-36
Set number of points to record	SxSetRecExtClkPoints	3-37
Query number of points to record	SxGetRecExtClkPoints	3-37
Set recording interval mode	SxSetRecIntervalMode	3-37
Query recording interval mode	SxGetRecIntervalMode	3-37
Set recording interval	SxSetRecInterval	3-38
Query recording interval	SxGetRecInterval	3-38
Set recording interval points	SxSetRecExtClkInterval	3-38
Query recording interval points	SxGetRecExtClkInterval	3-38
Set record count	SxSetRecTimes	3-39
Query record count	SxGetRecTimes	3-39
Set recording destination folder (PC)	SxSetRecFileFolder	3-39
Query recording destination folder (PC)	SxGetRecFileFolder	3-39
Set auto naming	SxSetRecFileAutoNaming	3-40
Query auto naming	SxGetRecFileAutoNaming	3-40
Set file name	SxSetRecFileName	3-40
Query file name	SxGetRecFileName	3-40
Set file order	SxSetRecFileOrder	3-41
Query file order	SxGetRecFileOrder	3-41
Set file count limit	SxSetRecCyclicFiles	3-41
Query file count limit	SxGetRecCyclicFiles	3-41
Set comment	SxSetRecFileComment	3-42
Query comment	SxGetRecFileComment	3-42
Get info. about file being recorded (PC)	SxGetRecCurrentFileInfo	3-42
Get info. about recorded files (PC)	SxGetRecLastFileInfo	3-42

Measurement Controls

Function	Name	Page
Start measurement	SxAcqStart	3-43
Stop measurement	SxAcqStop	3-43
Execute latch (in Free Run mode)	SxAcqLatch	3-43
	SxAcqLatchL	
	SxAcqLatchD	
Confirm measure/save operation	SxIsRun	3-44
Execute manual trigger	SxExecManualTrig	3-44

Auto Recording Controls

Function	Name	Page
Start recording	SxRecStart	3-45
Stop recording	SxRecStop	3-45
Divide file	SxRecDevide	3-45

Acquisition and Deletion of Measured Data

Function	Name	Page
Get acquisition data information (constants for converting physical values)	SxGetChannelInfo	3-46
Get latch interval sample points	SxGetLatchLength	3-46
Get latest acquisition number	SxGetLatestAcqNo	3-46
	SxGetLatestAcqNoL	
	SxGetLatestAcqNoD	
Get waveform data	SxGetAcqData	3-47
	SxGetAcqDataL	
	SxGetAcqDataD	
Get waveform data segment	SxGetAcqDataEx	3-48
	SxGetAcqDataExL	
	SxGetAcqDataExD	
Get data acquisition time	SxGetAcqTime	3-49
	SxGetAcqTimeL	
	SxGetAcqTimeD	
Get measuring group phase difference	SxGetAcqDelay	3-50
	SxGetAcqDelayL	
	SxGetAcqDelayD	
Create WDF file	SxSaveAcqData	3-50
	SxSaveAcqDataL	
	SxSaveAcqDataD	
Create WDF file (from waveform data segment)	SxSaveAcqDataEx	3-51
	SxSaveAcqDataExL	
	SxSaveAcqDataExD	
Get instantaneous values	SxGetCurrentData	3-52
Delete waveform data	SxClearAcqData	3-52

Setup Data Access

Function	Name	Page
Save setup data	SxSaveSetup	3-53
Load setup data	SxLoadSetup	3-53
Initialize setup data	SxInitSetup	3-53

System-Related Functions

Function	Name	Page
Execute calibration	SxExecCal	3-54
Execute self test	SxExecSleftest	3-54
Get unit's CPU temperature	SxGetCpuTemperature	3-54
Set unit's clock	SxSetSystemClock	3-55
Query unit's clock	SxGetSystemClock	3-55
Set DHCP	SxSetEthernetDHCP	3-56
Query DHCP	SxGetEthernetDHCP	3-56
Set IP address	SxSetEthernetIP	3-56
Query IP address	SxGetEthernetIP	3-56
Set subnet mask	SxSetEthernetNetMask	3-57
Query subnet address	SxGetEthernetNetMask	3-57
Set default gateway	SxSetEthernetGateway	3-57
Query default gateway	SxGetEthernetGateway	3-57
Set SNTP	SxSetSNTP	3-58
Query SNTP	SxGetSNTP	3-58
Set key lock	SxSetKeyLock	3-59
Query key lock	SxGetKeyLock	3-59
Get error code	SxGetError	3-59

Internal Media Operations

Function	Name	Page
Query number of drives	SxFileGetDriveNum	3-60
Query drive information	SxFileGetDriveInfo	3-60
Set current drive	SxFileSetCurrentDrive	3-60
Query current drive	SxFileGetCurrentDrive	3-61
Set current directory	SxFileChDir	3-61
Query current directory	SxFileCwDir	3-61
Create subdirectory	SxFileMkDir	3-62
Delete subdirectory	SxFileRmDir	3-62
Get number of files	SxFileGetFileNum	3-62
Get file information.	SxFileGetFileInfo	3-63
Delete file	SxFileDelete	3-63
Get file	SxFileGet	3-64
Create file	SxFilePut	3-64
	SxFilePutM	

Debugging

Function	Name	Page
Set trace mode	SxTraceSetMode	3-63
Query trace mode	SxTraceGetMode	3-64
Output trace	SxTracePrint	3-64
Reset performance timer	SxResetTimer	3-64
Get performance timer	SxGetTimer	3-65

3.2 Initialization and Ending

Initialization

int SxInit(HWND hWnd, int wire, char *option, SX_HNDL_COMM *hComm);

ERR Init(WIRE wire, string option, ref HNDL hComm);

Description

Executing this function initializes the network, initiates the search for connected units, initializes the API runtime environment, and returns the communication handle of the specified network. When an application program uses SxAPI, it must execute this function first.

If an application executes this function, it must then execute SxExit() before it closes.

Parameters

hWnd Specifies the window to send a message to when an SRQ is received. If 0 is specified, the message will be sent to the main window. This parameter is invalid in the .NET interface.

wire Sets the type of network to connect to.
SX_WIRE_USB: USB
SX_WIRE_LAN: Ethernet

option Specifies an optional network text string.
This parameter is invalid if the connection type is USB.
If the connection type is Ethernet, this parameter specifies the subnet mask to use to search for units.

Example 1: ""(Null) All units within the subnet are searched.
Example 2: "255.255.255.0" Searches for units with the specified subnet mask.
Example 3: "192.168.21.3" Only searches for the unit with the specified address.

hComm Specifies where to store the initialized communication handle. If there is an error, 0 will be stored.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Exit

int SxExit(SX_HNDL_COMM hComm);

ERR Exit(HNDL hComm);

Description

Executing this function closes the API runtime environment and the network drivers for the network specified by hComm. Always execute this function at the end of an application program.

Parameters

hComm A communication handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Re-Search

int SxReSearch(SX_HNDL_COMM hComm);

ERR ReSearch(HNDL hComm);

Description

Re-searches for units on the network specified by hComm. When this function is executed, previously retrieved handles with lower precedence than the communication handle (unit group handles, unit handles, module handles, channel handles, and measuring group handles) are released. So, when this function is executed, it is necessary to re-retrieve those low precedence handles. Be aware that the use of previously retrieved low-precedence handles will produce unexpected results.

Parameters

hComm	A communication handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.3 Retrieving Device Information

Get Unit Group Information

int SxGetGroupInfo(SX_HNDL_COMM | SX_HNDL_GROUP hAny, int groupNo,SX_INFO_GROUP *groupinfo);

ERR GetGroupInfo(HNDL hAny, int groupNo, ref INFO_GROUP groupinfo);

Description

If hAny is a communication handle, this function retrieves the unit group information of the unit group specified by groupNo.

If hAny is a unit group handle, this function retrieves the unit group information of the group with that handle. In this case, the groupNo parameter is invalid.

Parameters

hAny	A communication handle or unit group handle
groupNo	A group number (0 to 15). This parameter is disabled when a unit group handle is specified for hAny.
groupinfo	Specifies where to store the unit group information.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Number of Units

int SxGetUnitNum(SX_HNDL_COMM hComm, int groupNo, int *unitNum);

ERR GetUnitNum(HNDL hComm, int groupNo, ref int unitNum);

Description

Retrieves the number of units in the unit group specified by groupNo.

Parameters

hComm	A communication handle
groupNo	A group number (0 to 15)
unitNum	Specifies where to store the unit number

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Unit Information

int SxGetUnitInfo(SX_HHDL_COMM | SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int groupNo, int unitNo, SX_INFO_UNIT *unitInfo);

ERR GetUnitInfo(HNDL hComm, int groupNo, int unitNo, ref INFO_UNIT unitInfo);

Description

If hAny is a communication handle, this function retrieves the unit information of the unit specified by unitNo that is in the unit group specified by groupNo.
If hAny is a unit group handle, groupNo is ignored and this function retrieves the unit information list of the unit by unitNo.
If hAny is a unit handle, this function retrieves the unit information list of the unit with that handle, and groupNo and unitNo are ignored.
An error is returned if there are two or more units that meet the conditions specified.

Parameters

hAny	Any kind of handle
groupNo	A group number (0 to 15)
unitNo	A unit number (0 to 7)
unitInfo	Specifies where to store the unit information

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetUnitInfoS(SX_HNDL_COMM hComm, int groupNo, int index, SX_INFO_UNIT *unitInfo);

ERR GetUnitInfoS(HNDL hComm, int groupNo, int index, ref INFO_UNIT unitInfo);

Description

Retrieves the unit information of the unit in the group specified by groupNo with the specified index (the order in which the units were found, starting with 0). The maximum number that can be set for the index is equal to the number of units acquired with SxGetUnitNum()–1.

Parameters

hComm	A communication handle
groupNo	A group number (0 to 15)
index	An index number (from 0 to the number of units − 1)
unitInfo	Specifies where to store the unit information

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Module Information

int SxGetModInfo(SX_HNDL_MOD hMod, SX_INFO_MOD *modInfo);

ERR GetModInfo(HNDL hMod, ref INFO_MOD modInfo);

Description

Retrieves the module information of the module specified by hMod.

Parameters

hMod	A module handle
modInfo	Specifies where to store the module information

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.4 Opening and Closing Handles

Open Unit Group

int SxOpenGroup(SX_HNDL_COMM hComm, int groupNo, SX_HNDL_GROUP *hGrp);

int SxOpenGroupEx(SX_HNDL_COMM hComm, int groupNo, SX_HNDL_GROUP *hGrp, int comm_tout, int alive_tout);

ERR OpenGroup(HNDL hComm, int groupNo, ref HNDL hGrp);

ERR OpenGroupEx(HNDL hComm, int groupNo, ref HNDL hGrp, int comm_tout, int alive_tout);

Description

Retrieves the handle of the unit group specified by groupNo with the communication handle specified by hComm. Returns an error if the specified unit group does not exist, or if it has an invalid ID.

You can specify the communication timeout and connection timeout with SxOpenGroupEx().

If you use SxOpenGroup(), the communication timeout is set at 5 s, and the connection timeout is set at 10 s.

Be sure to close the unit group handles retrieved with these functions with SxCloseGroup() before closing the communication handle with SxExit().

Parameters

hComm	A communication handle
groupNo	A unit group number (0 to 15)
hGrp	Specifies where to store the unit group handle.
comm_tout	Communication timeout in seconds.
alive_tout	Connection timeout in seconds.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Close Unit Group

int SxCloseGroup(SX_HNDL_GROUP hGrp);

ERR CloseGroup(HNDL hGrp);

Description

Closes the unit group specified by hGrp.

Be sure to close the unit group handles retrieved with SxOpenGroup() or SxOpenGroupEx() with this function before closing the communication handle with SxExit().

Parameters

hGrp	A unit group handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Open Unit

int SxOpenUnit(SX_HNDL_COMM hComm, char *address, SX_HNDL_UNIT *hUnit);

int SxOpenUnitEx(SX_HNDL_COMM hComm, char *address, SX_HNDL_UNIT *hUnit, int comm_tout, int alive_tout);

ERR OpenUnit(HNDL hComm, string address, ref HNDL hUnit);

ERR OpenUnitEx(HNDL hComm, string address, ref HNDL hUnit, int comm_tout, int alive_tout);

Description

Retrieves the handle of a unit with the specified address in the network specified by hComm. An error is returned if the unit with the specified address does not exist.

You can specify the communication timeout and connection timeout with SxOpenUnitEx(). If you use SxOpenUnit(), the communication timeout is set at 5 s, and the connection timeout is set at 10 s.

To control measurement and auto recording, a unit group handle is necessary. Use a function such as SxOpenGroup to open a unit group.

Be sure to close the unit group handles acquired with these functions with SxCloseUnit() before closing the communication handle with SxExit().

Parameters

hComm	A communication handle
address	An address
	Use a serial number for USB (e.g., "12A456789")
	Use an IP address for Ethernet (e.g., "192.168.21.3")
hUnit	Specifies where to store the unit handle.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Close Unit

int SxCloseUnit(SX_HNDL_UNIT hUnit);

ERR CloseUnit(HNDL hUnit);

Description

Closes the unit specified by hUnit.

Be sure to close the unit handles acquired with SxOpenUnit() or SxOpenUnitEx() with this function before closing the communication handle with SxExit().

Parameters

hUnit	A unit handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.5 Handle Acquisition

Get Communication Handle

SX_HNDL_COMM SxMyCommHndl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny);

HNDL MyCommHndl(HNDL hAny);

Description

Returns the communication handle of the network that contains the object specified by hAny.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is a communication handle. If the function fails, the return value is zero.

Get Unit Group Handle

SX_HNDL_GROUP SxMyGrpHndl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny);

HNDL MyGrpHndl(HNDL hAny);

Description

Returns the unit group handle that contains the object specified by hAny.
If hAny is a unit group, the function returns hAny.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is a unit group handle. If the function fails, the return value is zero.

Get Unit Handle

SX_HNDL_UNIT SxMyUnitHndl(SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny);

HNDL MyUnitHndl(HNDL hAny);

Description

Returns the handle of the unit that contains the object specified by hAny.
If hAny is a unit, the function returns hAny.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is a unit handle. If the function fails, the return value is zero.

Get Unit Handle (By Unit Number)

SX_HNDL_UNIT SxUnitHndl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int unitNo);

HNDL UnitHndl(HNDL hAny, int unitNo);

Description

Returns the handle of the unit specified by unitNo in the unit group that contains the object specified by hAny.
If hAny is a unit, the function returns hAny. (The unitNo parameter is invalid.)
If hAny is a module or channel, the function returns the handle of the unit that contains hAny. (The unitNo parameter is invalid.)

Parameters

hAny	Any kind of handle
unitNo	A unit number (0 to 7)

Return Value

If the function succeeds, the return value is a unit handle. If the function fails, the return value is zero.

Get Measuring Group Handle

SX_HNDL_MEASGRP SxMyMeasgrpHndl(SX_HNDL_MOD | SX_HNDL_CH hAny);

HNDL MyMeasgrpHndl(HNDL hAny);

Description

Returns the handle of the measuring group that contains the object specified by hAny.

Parameters

hAny	Any kind of handle

Return Value

If the function succeeds, the return value is a unit handle. If the function fails, the return value is zero.

Get Measuring Group Handle (By Measuring Group Number)

SX_HNDL_MEASGRP SxMeasgrpHndl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int sampleNo);

HNDL MeasgrpHndl(HNDL hAny, int sampleNo);

Description

Returns the handle of the measuring group specified by sampleNo in the unit group that contains the object specified by hAny.
If hAny is a measuring group, the function returns hAny. (The sampleNo parameter is invalid.)
If hAny is a module or channel, the function returns the measuring group that contains hAny. (The sampleNo parameter is invalid.)

Parameters

hAny	Any kind of handle
sampleNo	A measuring group number (0 to 3)

Return Value

If the function succeeds, the return value is a measuring group handle. If the function fails, the return value is zero.

Get Module Handle

SX_HNDL_MOD SxMyModHndl(SX_HNDL_MOD | SX_HNDL_CH hAny);

HNDL MyModHndl(HNDL hAny);

Description

If hAny is a channel, the function returns the handle of the module that contains hAny.
If hAny is a module, the function returns hAny.

Parameters

hAny	Any kind of handle

Return Value

If the function succeeds, the return value is a module handle. If the function fails, the return value is zero.

Get Module Handle (By Module Number)

SX_HNDL_MOD SxModHndl(SX_HNDL_GROUP | SX_HNDL_UNIT hAny | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int moduleNo);

HNDL ModHndl(HNDL hAny, int moduleNo);

Description

Returns the handle of the module specified by moduleNo in the unit that contains the object specified by hAny. If hAny is a unit group or measuring group, specify the moduleNo with a sequence number starting with unit 0.
If hAny is a module, the function returns hAny. (The moduleNo parameter is invalid.)
If hAny is a channel, the function returns the handle of the module that contains hAny. (The moduleNo parameter is invalid).

Parameters

hAny	Any kind of handle
moduleNo	A module number (if hAny is SX_HNDL_GROUP, use a sequence number starting with unit 0.)

Return Value

If the function succeeds, the return value is a module handle. If the function fails, the return value is zero.

Get Channel Handle

SX_HNDL_CH SxChHndl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int channelNo);

HNDL ChHndl(HNDL hAny, int channelNo);

Description

Retrieves the handle of the channel specified by channelNo that belongs to the object specified by hAny.
If hAny is a unit group, specify the channel with a sequence number starting with unit 0.
If hAny is a unit, specify the channel with the unit's logic channel number.
If hAny is a module, specify the channel with the module's channel number, based on its order with the first channel being 0.
If hAny is a channel, the function returns hAny. (The channelNo parameter is invalid.)
If hAny is a measuring group, specify the channel with a sequence number starting with unit 0 (leaving out channels in other measuring groups).

Parameters

hAny	Any kind of handle
channelNo	A channel number

Return Value

If the function succeeds, the return value is a channel handle. If the function fails, the return value is zero.

3.6 Value Retrieval

Get Channel Number

int SxChNo(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny);

int ChNo(HNDL hAny);

Description

Returns the logic channel number of the first channel in the object specified by hAny.
The function returns 0 if hAny is a unit group.
If hAny is a unit, the function returns the logic channel number of the first channel in that unit.
If hAny is a module, the function returns the logic channel number of the first channel in that module.
If hAny is a channel, the function returns the logic channel number of the channel.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is a logic channel number. If the function fails, the return value is –1.

Get Module Number

int SxModNo(SX_HNDL_MOD | SX_HNDL_CH hAny);

int ModNo(HNDL hAny);

Description

Returns the slot number of the module that contains the object specified by hAny.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is a slot number. If the function fails, the return value is –1.

Get Unit Number

int SxUnitNo(SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny);

int UnitNo(HNDL hAny);

Description

Returns the unit number of the unit that contains the object specified by hAny.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is a unit number. If the function fails, the return value is –1.

Get Unit Group Number

int SxGrpNo(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny);

int GrpNo(HNDL hAny);

Description

Returns the unit group number of the group that contains the object specified by hAny.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is a unit number. If the function fails, the return value is –1.

Get Measuring Group Number

int SxMeasgrpNo(SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny);

int MeasgrpNo(HNDL hAny);

Description

Returns the measuring group number of the measuring group that contains the object specified by hAny.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is a measuring group number. If the function fails, the return value is –1.

Get Channel Number

int SxChNum(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny);

int ChNum(HNDL hAny);

Description

Returns the total number of channels in the object specified by hAny.
If hAny is a channel, the function returns 1.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is the number of channels. If the function fails, the return value is –1.

Get Module Number

int SxModNum(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_MEASGRP hAny);

int ModNum(HNDL hAny);

Description

Returns the total number of modules in the object specified by hAny.
If hAny is a module, the function returns 1.
If hAny is a measuring group, the function returns the total number of modules in the group.

Parameters

hAny Any kind of handle

Return Value

If the function succeeds, the return value is the number of modules. If the function fails, the return value is –1.

Get Unit Number

int SxUnitNum (SX_HNDL_GROUP hGrp);

int UnitNum(HNDL hGrp);

Description

Returns the number of units in the unit group specified by hGrp.

Parameters

hGrp A unit group handle

Return Value

If the function succeeds, the return value is the number of units. If the function fails, the return value is –1.

3.7 Measuring Group Settings

Setup Measuring Group

int SxSetupMeasgrp(SX_HNDL_MOD hMod, int measgrpNo);

ERR SetupMeasgrp(HNDL hMod, int measgrpNo);

Description

Assigns the module specified by hMod to the measuring group specified by measgrpNo.

Parameters

hMod	A module handle
measgrpNo	A measuring group number (0 to 3)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.8 Communication Command Controls

Send Command

int SxSetControl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, char *msg);

ERR SetControl(HNDL hAny, string msg);

Description

Sends a program message to the unit specified by hAny, waits for processing to finish, and then exits from the function.

If hAny is a unit group or measuring group, the message will be sent to all units in the group. To send multiple commands at once, separate each command with a semicolon.

Even if the program message contains a query (a command that ends with a question mark), the response to the command will be discarded.

For information about what commands can be sent, see chapter 4.

If <ch> or <mo> is contained in the command string, the actual string that will be sent will follow the pattern of the examples shown below.

When the Command String Contains <ch>

If hAny is a unit group,
the command will be copied for as many times as there are channels in the unit group, and <ch> will be replaced by the physical channel number.

Example:

Command string: `"CHAN<ch>:RANG 10.0"`

String sent to unit 0:

`"CHAN1:RANG 10.0;CHAN2:RANG 10.0; ... ;CHAN16:RANG 10.0"`

String sent to unit 1:

`"CHAN17:RANG 10.0;CHAN18:RANG 10.0; ... ;CHAN32:RANG 10.0"`

If hAny is a unit,
the command will be copied for as many times as there are channels in the unit, and <ch> will be replaced by the physical channel number.

Example:

Command string: `"CHAN<ch>:RANG 10.0"`

String sent to the unit:
```
"CHAN17:RANG 10.0;CHAN18:RANG 10.0;
... ;CHAN32:RANG 10.0"
```

If hAny is a module,
the command will be copied for as many times as there are channels in the module, and <ch> will be replaced by the physical channel number.

Example:

Command string: `"CHAN<ch>:RANG 10.0"`

String sent to the unit: `"CHAN5:RANG 10.0;CHAN6 10.0"`

If hAny is a channel,
<ch> will be replaced by the channel's physical channel number.

Example:

Command string: `"CHAN<ch>:RANG 10.0"`

String sent to the unit: `"CHAN11:RANG 10.0"`

If hAny is a measuring group,
the command will be copied for as many times as there are channels in the measuring group, and <ch> will be replaced by the physical channel number.

Example:

Command string:	"CHAN<ch>:RANG 10.0"
String sent to unit 0:	"CHAN3:RANG 10.0;CHAN4:RANG 10.0;"
String sent to unit 1:	"CHAN19:RANG 10.0;CHAN20:RANG 10.0;"

When the Command String Contains <mo>

If hAny is a unit group,
the command will be copied for as many times as there are modules in the unit group, and <mo> will be replaced by the physical slot number.

Example:

Command string:	":TIM:MODU<mo>:GROU 4"
String sent to unit 0:	":TIM:MODU1:GROU 4;:TIM:MODU2:GROU 4; ... ;:TIM:MODU8:GROU 4"
String sent to unit 1:	":TIM:MODU9:GROU 4; ... :TIM:MODU15:GROU 4;:TIM:MODU16:GROU 4"

If hAny is a unit,
the command will be copied for as many times as there are modules in the unit, and <mo> will be replaced by the physical slot number.

Example:

Command string:	":TIM:MODU<mo>:GROU 4"
String sent to the unit:	":TIM:MODU1:GROU 4;:TIM:MODU2:GROU 4; ... ;:TIM:MODU8:GROU 4"

If hAny is a module,
<mo> will be replaced by the module's physical slot number.

Example:

Command string:	":TIM:MODU<mo>:GROU 4"
String sent to the unit:	":TIM:MODU3:GROU 4"

If hAny is a channel,
<mo> will be replaced by the physical slot number of the module that contains the channel.

Example:

Command string:	":TIM:MODU<mo>:GROU 4"
String sent to the unit:	":TIM:MODU3:GROU 4"

If hAny is a measuring group,
the command will be copied for as many times as there are modules in the measuring group, and <mo> will be replaced by the physical slot number.

Example:

Command string:	":TIM:MODU<mo>:GROU 4"
String sent to unit 0:	":TIM:MODU1:GROU 4;:TIM:MODU2:GROU 4; ... ;:TIM:MODU8:GROU 4"
String sent to unit 1:	":TIM:MODU9:GROU 4; ... :TIM:MODU15:GROU 4;:TIM:MODU16:GROU 4"

Parameters

hAny	Any kind of handle
msg	The initial address of the program message to be sent

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Send Binary Data

int SxSetControlBinary(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, char *msg, char *buf, int len);

ERR SetControlBinary(HNDL hAny, string msg, any[] buf, int len);

ERR SetControlBinary(HNDL hAny, string msg, any[] buf);

Description

Sends a program message with binary parameters to the unit specified by hAny, waits for processing to finish, and then exits from the function.

If hAny is a unit group or measuring group, the message will be sent to all units in the group. This function cannot send multiple commands. Also, the function will not receive responses to queries (commands that end with question marks).

For information about what commands can be sent, see chapter 4.

If the command string contains `<ch>` or "<mo>," it will be copied and altered in the same ways as with SxSetControl().

Parameters

hAny	Any kind of handle
msg	The initial address of the header of the program message to be sent
buf	The initial address of the binary parameters to be sent
len	The length of the parameters to be sent, specified in bytes

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Send and Receive Command

int SxGetControl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, char *msg, char *buf, int blen, int *rlen);

ERR GetControl(HNDL hAny, string req, ref string rep, int blen, ref int rlen);

ERR GetControl(HNDL hAny, string req, ref string rep, int blen);

ERR GetControl(HNDL hAny, string req, ref string rep);

Description

Sends a program message to the unit specified by hAny, receives a response, and then exits from the function.

If hAny is a unit group or measuring group, the message will be sent to all units in the group. To send multiple commands, separate each command with a semicolon.

If there are no queries (commands that end with question marks) in the program message, a timeout error will result.

For information about what commands can be sent, see chapter 4.

If the command string contains `<ch>` or "<mo>," it will be copied and altered in the same ways as with SxSetControl().

Parameters

hAny	Any kind of handle
msg	The initial address of the program message to be sent
buf	Where to store the response message
blen	The size of the area for storing the response message, specified in bytes
rlen	Where to store the length (in bytes) of the response message string

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Send and Receive Command (Binary Data Reception)

int SxGetControlBinary(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, char *msg, char *buf, int blen, int *rlen);

ERR GetControlBinary(HNDL hAny, string req, ref any[] buf, int blen, ref int rlen);

ERR GetControlBinary(HNDL hAny, string req, ref any[] buf, ref int rlen);

Description

Sends a program message to the unit specified by hAny, receives a response, and then exits from the function. The received response message's parameters must be in binary format.

If hAny is a unit group or measuring group, the message will be sent to all units in the group. To send multiple commands, separate each command with a semicolon. You cannot send multiple queries.

If there are no queries (commands that end with question marks) in the program message, a timeout error will result.

For information about what commands can be sent, see chapter 4.

If the command string contains `<ch>` or "<mo>," it will be copied and altered in the same ways as with SxSetControl().

Parameters

hAny	Any kind of handle
msg	The initial address of the program message to be sent
buf	Where to store the response message (that contains binary parameters)
blen	The size of the area for storing the response message, specified in bytes
rlen	Where to store the length of the response message, specified in bytes

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Received Parameters (Copy to Specified Buffer)

int SxGetParam(char *rep, int pos, int idx, char *buf, int blen);

int SxGetParamStr(char *rep, int pos, int idx, char *buf, int blen);

ERR GetParam(string rep, int pos, int idx, ref string buf, int blen);

ERR GetParam(string rep, int pos, int idx, ref string buf);

ERR GetParamStr(string rep, int pos, int idx, ref string buf, int blen);

ERR GetParamStr(string rep, int pos, int idx, ref string buf);

Description

Extracts the parameter string with the specified number from the received message string and stores it in buf.

The SxGetParamStr() function removes the double quotation marks from the beginning and end of a parameter string that starts with a double quotation mark before storing it.

If the parameter with the specified number does not exist, the function will return a nonzero error code.

Parameters

rep	The initial address of the received message string.
pos	The command number (starting with zero). Commands are separated with a semicolon.
idx	The element number (starting with zero). Elements are pieces of text separated with commas.
buf	Where to store the extracted parameter string
blen	The size of the area for storing the extracted parameter string.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Received Parameters (Get Location)

char* SxGetParamPos(char *rep, int pos, int idx);

Description

Returns the location of the parameter string with the specified number in the received message string.

This function can only be used with VC++.

Parameters

rep	The initial address of the received message string.
pos	The command number (starting with zero). Commands are separated with a semicolon.
idx	The element number (starting with zero). Elements are pieces of text separated with commas.

Return Value

The function returns the initial address of the specified parameter text string. If there is an error, the function returns NULL.

3.9 Event Controls

Create and Enable Event Handler

int SxCreateEvent(SX_HNDL_GROUP hGrp, HWND hWnd, ULONG enable);

ERR CreateEvent(HNDL hGrp, uint enable);

Description

Allows a message to be sent when the unit group specified by hGrp creates an event.
Enables events specified with a 1 bit in the enable parameter.
hWnd specifies the handle of the window to which the message will be sent.
hWnd is invalid in the .NET interface.
If hWnd is NULL, the message will be sent to the main window.
The window message name is “WM_YOKOGAWA_TM_SX_EVENT.”
The window message ID can be acquired with this function: RegisterWindowMessage(” WM_YOKOGAWA_TM_SX_EVENT”).
The window message’s WPARAM is the event source’s unit handle. LPARAM is the event’s bit pattern.
Be sure to close the event handlers created with this function with SxDeleteEvent() before closing the communication handle with SxExit().
For details about event handling, see chapter 2.

Parameters

hGrp	A unit group handle
hWnd	The handle of the window to which the message will be sent (If the value is NULL, the message will be sent to the main window.)
enable	The event enabling bit pattern (enable with 1, disable with 0)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Delete Event Handler

int SxDeleteEvent(SX_HNDL_GROUP hGrp);

ERR DeleteEvent(HNDL hGrp);

Description

Closes the event handler of the unit group specified by hGrp.
Be sure to use this function to close the event handlers created with SxCreateEvent() before closing the communication handle with SxExit().

Parameters

SX_HNDL_GROUP hGrp	A unit group handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Enable Next Trigger and Related Event Notification

int SxEnableNextTrig(SX_HNDL_GROUP | SX_HNDL_UNIT hAny);

ERR EnableNextTrig(HNDL hAny);

Description

Enables trigger detection and TRG_START and TRIG_END event raising by the master unit, if hAny is a unit group, or by the unit specified by hAny, if hAny is a unit.
So that units do not produce communication and PC processing problems by sending multiple TRG_START and TRIG_END events in Single (N) mode, events after the first event will not be raised until this function is executed. To receive TRG_START and TRIG_END events after the first event, this function must be executed after each TRG_START or TRIG_END event is received to allow the raising of the next event.
In Normal mode, the SL1000 begins detection of the next trigger after recieving permission from the PC. So it is necessary to use this function to enable trigger detection in order to enable triggering after the first trigger.

Parameters

hAny Any kind of handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.10 Measurement Condition Settings and Queries

Query or Set Measurement On/Off

int SxSetAcqSwitch(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int acqSW);

ERR SetAcqSwitch(HNDL hAny, BOOL acqSW);

Description

Turns the measurement of the channels contained in the object specified by hAny on or off.

Parameters

hAny	Any kind of handle
acqSW	0: Off 1: On

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetAcqSwitch(SX_HNDL_CH hCh, int *acqSW);

ERR GetAcqSwitch(HNDL hCh, ref BOOL acqSW);

Description

Queries the on/off status of the measurement of the channel specified by hCh.

Parameters

hCh	A channel handle
acqSW	Where to store the measurement on/off information 0: Off 1: On

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Auto Recording On/Off

int SxSetRecSwitch(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int recSW);

ERR SetRecSwitch(HNDL hAny, BOOL recSW);

Description

Turns the auto recording of the channels contained in the object specified by hAny on or off.

Parameters

hAny	Any kind of handle
recSW	0: Off 1: On

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecSwitch(SX_HNDL_CH hCh, int *recSW);

ERR GetRecSwitch(HNDL hCh, ref BOOL recSW);

Description

Queries the on/off status of the auto recording of the channel specified by hCh.

Parameters

hCh	A channel handle
recSW	Where to store the auto recording on/off information 0: Off 1: On

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Measuring Mode

int SxSetAcqMode(SX_HNDL_GROUP hGrp, int acqMode);

ERR SetAcqMode(HNDL hGrp, ACQMODE acqMode);

Description

Sets the measuring mode of the unit group specified by hGrp to the mode specified by acqMode.

The acquisition mode is fixed at Normal. You cannot select the Envelope or Box Average modes.

Parameters

hGrp	A unit group handle
acqMode	SX_ACQ_FREE: Free run mode SX_ACQ_TRIG: Trigger mode

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetAcqMode(SX_HNDL_GROUP hGrp, int *acqMode);

ERR GetAcqMode(HNDL hGrp, ref ACQMODE acqMode);

Description

Queries the measuring mode of the measuring group specified by hGrp.

Parameters

hGrp	A unit group handle
acqMode	Where to store the measuring mode information SX_ACQ_FREE: Free run mode SX_ACQ_TRIG: Trigger mode

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Sampling Clock

int SxSetClockSource(SX_HNDL_GROUP hGrp, int clockSource);

ERR SetClockSource(HNDL hGrp, CLOCKSOURCE clockSource);

Description

Sets the source of the sampling clock of the unit group specified by hGrp to the source specified by clock.

Parameters

hGrp	A unit group handle
clockSource	SX_CLK_INT: The internal clock
	SX_CLK_EXT: An external clock

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetClockSource(SX_HNDL_GROUP hGrp, int *clockSource);

ERR GetClockSource(HNDL hAny, ref CLOCKSOURCE clockSource);

Description

Queries the source of the sampling clock of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle
clockSource	Where to store the clock source information
	SX_CLK_INT: The internal clock
	SX_CLK_EXT: An external clock

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Sample Rate

int SxSetSamplingRate(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MEASGRP hAny, double smplRate);

ERR SetSamplingRate(HNDL hAny, double smplRate);

Description

Sets the sample rate of the measuring group specified by hAny to the sample rate specified by smplRate.

If hAny is a unit or unit group, all of the measuring groups that hAny contains will be set to the same sample rate.

Parameters

hAny	Any kind of handle
smplRate	The sample rate frequency (in Hz)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetSamplingRate(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MEASGRP hAny, double *smplRate);

ERR GetSamplingRate(HNDL hAny, ref double smplRate);

Description

Queries the sample rate of the measuring group specified by hAny.

Parameters

hAny	Any kind of handle
smplRate	Where to store the sample rate frequency (in Hz)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Sample Interval

int SxSetSamplingInterval(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MEASGRP hAny, double smplInterval);

ERR SetSamplingInterval(HNDL hAny, double smplInterval);

Description

Changes the sample rate of the measuring group specified by hAny to the inverse of the sample interval specified by smplInterval.
If hAny is a unit or unit group, all of the measuring groups that hAny contains will be set to the same sample rate.
The smplRate and smplInterval information for the measuring groups whose values are changed will be updated simultaneously.

Parameters

hAny	Any kind of handle
smplRate	The sample interval (in seconds)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetSamplingInterval(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MEASGRP hAny, double *smplInterval);

ERR GetSamplingInterval(HNDL hAny, ref double smplInterval);

Description

Queries the inverse of the sample rate of the measuring group specified by hAny.

Parameters

hAny	Any kind of handle
smplInterval	Where to store the sample interval information (in seconds)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Sample Points

int SxSetAcqLength(SX_HNDL_GROUP hGrp, int len);

ERR SetAcqLength(HNDL hGrp, int len);

Description

Sets the number of measurement points of the unit group specified by hGrp to the length specified by len. Specify the number of measurement points with the number of measurement group 1 measurement points.

Parameters

hGrp	A unit group handle
len	The number of measurement points

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetAcqLength(SX_HNDL_GROUP hGrp, int *len);

ERR GetAcqLength(HNDL hGrq, ref int len);

Description

Queries thenumber of measurement points of the unit group specified by hGrp. The number of measurement points is specified with the number of measurement group 1 measurement points.

Parameters

hGrp	A unit group handle
len	Where to store the number of measurement points

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Sampling Time

int SxSetAcqSpan(SX_HNDL_GROUP hGrp, double second);

ERR SetAcqSpan(HNDL hGrp, double second);

Description

Sets the measuring time of the unit group specified by hGrp to the number of seconds specified by the second parameter.
This setting is invalid if an external sampling clock is being used.

Parameters

hGrp	A unit group handle
second	The measuring time in seconds

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetAcqSpan(SX_HNDL_GROUP hGrp, double *second);

ERR GetAcqSpan(HNDL hGrp, ref double second);

Description

Queries the measuring time (in seconds) of the unit group specified by hGrp. The measuring time is specified with the number of measurement group 1 measuring time.

Parameters

hGrp	A unit group handle
second	The measuring time in seconds

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Trigger Mode

int SxSetTrigMode(SX_HNDL_GROUP hGrp, int trigMode);

ERR SetTrigMode(HNDL hGrp, TRIGMODE trigMode);

Description

Sets the trigger mode of the measuring group specified by hGrp to the mode specified by trigMode.

Parameters

hGrp	A unit group handle
trigMode	SX_TRIG_NORMAL: Normal mode SX_TRIG_SINGLE: Single mode SX_TRIG_NSINGLE: Single (N) mode

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetTrigMode(SX_HNDL_GROUP hGrp, int *trigMode);

ERR GetTrigMode(HNDL hGrq, ref TRIGMODE trigMode);

Description

Queries the trigger mode of the measuring group specified by hGrp.

Parameters

hGrp	A unit group handle
trigMode	Where to store the trigger mode SX_TRIG_NORMAL: Normal mode SX_TRIG_SINGLE: Single mode SX_TRIG_NSINGLE: Single (N) mode

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Pre-Trigger Position

int SxSetTrigPos(SX_HNDL_GROUP hGrp, double percent);

ERR SetTrigPos(HNDL hGrp, double percent);

Description

Sets the pre-trigger length of the unit group specified by hGrp to a percentage (specified by percent) of the total measuring time.
This setting is invalid if an external sampling clock is being used.

Parameters

hGrp	A unit group handle
percent	The pre-trigger length (as a percentage)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetTrigPos(SX_HNDL_GROUP hGrp, double *percent);

ERR GetTrigPos(HNDL hGrq, ref double percent);

Description

Queries the pre-trigger length as a percentage of the total measuring time of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle
percent	Where to store the pre-trigger length (as a percentage)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Pre-Trigger Points

int SxSetPretrigLength(SX_HNDL_GROUP hGrp, int points);

ERR SetPretrigLength(HNDL hGrp, int points);

Description

Sets the pre-trigger length of the unit group specified by hGrp to the specified number of measurement points. The length is specified with the number of measurement group 1 points that are equivalent to that length.

Parameters

hGrp	A unit group handle
points	The pre-trigger length (number of sample points)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

SxGetPretrigLength(SX_HNDL_GROUP hGrp, int *points);

ERR GetPretrigLength(HNDL hGrp, ref int points);

Description

Queries the pre-trigger length in sample points of unit group specified by hGrp (pre-trigger length based on measuring group 1).

Parameters

hGrp	A unit group handle
points	Where to store the pre-trigger length (number of sample points)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Trigger Count

int SxSetTrigCount(SX_HNDL_GROUP hGrp, int trigCount);

ERR SetTrigCount(HNDL hGrp, int trigCount);

Description

Sets the trigger count of the unit group specified by hGrp to the count specified by trigCount.
This function is invalid when the trigger mode is SX_TRIG_SINGLE.

Parameters

hGrp	A unit group handle
trigCount	The trigger count (0 indicates no limit)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetTrigCount(SX_HNDL_GROUP hGrp, int *trigCount);

ERR GetTrigCount(HNDL hGrp, ref int trigCount);

Description

Queries the trigger count of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle
trigCount	Where to store the trigger count (0 indicates no limit)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Channel Label

int SxSetChLabel(SX_HNDL_CH hCh, char *label);

ERR SetChLabel(HNDL hCh, string label);

Description

Sets the label of the channel specified by hCh to the string specified by the label parameter. Trying to set the label to a string longer than 7 characters will result in an error.

Parameters

hCh	A channel handle
label	The label character string (up to 8 characters including the null character)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetChLabel(SX_HNDL_CH hCh, char *label);

ERR GetChLabel(HNDL hCh, ref string label);

Description

Queries the label of the channel specified by hCh.
Prepare a buffer of 8 bytes or more for the label. If the label is 7 characters or less, a string that is terminated with the null character will be stored. If the label is 8 characters, it will not be terminated with the null character.

Parameters

hCh	A channel handle
label	Where to store the label string (8 bytes or more)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query Acquisition Data Capacity

int SxGetAcqCapacity(SX_HNDL_GROUP hGrp, int *size);

ERR GetAcqCapacity(HNDL hGrq, ref int size);

Description

Queries how much acquisition data can be stored in each channel in the unit group specified by hGrp. In Triggered mode, the acquisition data is expressed as the number of acquisition numbers. In Free Run mode, it is expressed as the number of sample points.

Parameters

hCh	A channel handle
size	Where to store the information about data storage capacity

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.11 Auto Recording Condition Settings and Queries

Query or Set Auto Recording Destination

int SxSetRecDest(SX_HNDL_GROUP hGrp, int dest);

ERR SetRecDest(HNDL hGrp, REC_DEST dest);

Description

Sets the auto recording destination of the unit group specified by hGrp to the destination specified by dest.

Parameters

hGrp	A unit group handle	
dest	SX_REC_DEST_PC:	The PC hard disk
	SX_REC_DEST_UNIT:	The unit's hard disk
	SX_REC_DEST_PC_UNIT	The PC and the unit's hard disks.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecDest(SX_HNDL_GROUP hGrp, int *dest);

ERR GetRecDest(HNDL hGrp, ref REC_DEST dest);

Description

Queries the auto recording destination of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle	
dest	Where to store the auto recording destination information	
	SX_REC_DEST_PC:	The PC hard disk
	SX_REC_DEST_UNIT:	The unit's hard disk
	SX_REC_DEST_PC_UNIT:	The PC and the unit's hard disks.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Recording Start Condition

int SxSetRecStartCond(SX_HNDL_GROUP hGrp, int starCond);

ERR SetRecStartCond(HNDL hGrp, REC_START recCond);

Description

Sets the recording start condition of the unit group specified by hGrp to the condition specified by startCond.

Parameters

hGrp	A unit group handle	
startCond	SX_REC_START_IMMIDIATE:	Start immediately
	SX_REC_START_TIME:	Start at a set time
	SX_REC_START_ALARM:	Start when an alarm occurs
	SX_REC_START_TRIG_RISE:	Start on the rising edge of an external trigger signal
	SX_REC_START_TRIG_FALL:	Start on the falling edge of an external trigger signal

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecStartCond(SX_HNDL_GROUP hGrp, int *startCond);

ERR GetRecStartCond(HNDL hGrp, ref REC_START recCond);

Description

Queries the recording start condition of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle
startCond	Where to store the recording start condition SX_REC_START_IMMIDIATE: Start immediately SX_REC_START_TIME: Start at a set time SX_REC_START_ALARM: Start when an alarm occurs SX_REC_START_TRIG_RISE: Start on the rising edge of an external trigger signal SX_REC_START_TRIG_FALL: Start on the falling edge of an external trigger signal

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Recording Start Time

int SxSetRecStartTime(SX_HNDL_GROUP hGrp, char *datetime);

ERR SetRecStartTime(HNDL hGrp, string datetime);

Description

Sets the recording start time of the unit group specified by hGrp to the date and time specified by datetime. This setting is only valid when the recording start condition is set to SX_REC_START_TIME.

Parameters

hGrp	A unit group handle
datetime	Where to store the date and time text string Format: "YYYY/MM/DD-hh:mm:ss" (19 characters + 1 null character) Example: Dec. 23, 2007, 15:22:33 > "2007/12/23-15:22:33"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecStartTime(SX_HNDL_GROUP hGrp, char *datetime);

ERR GetRecStartTime(HNDL hGrp, ref string datetime);

Description

Queries the recording start time of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle
datetime	Where to store the date and time text string (20 bytes or more) Format: "YYYY/MM/DD-hh:mm:ss" (19 characters + 1 null character) Example: Dec. 23, 2007, 15:22:33 > "2007/12/23-15:22:33"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Recording Stop Condition

int SxSetRecStopCond(SX_HNDL_GROUP hGrp, int stopCond);

ERR SetRecStopCond(HNDL hGrp, REC_STOP recCond);

Description

Sets the recording stop condition of the unit group specified by hGrp to the condition specified by stopCond.

Parameters

hGrp A unit group handle

stopCond
- SX_REC_STOP_CONTINUOUS: Continue until a recording stop command is received.
- SX_REC_STOP_TIME: Stop at the set time. (This setting is invalid when the recording start condition is set to SX_REC_START_ALARM).
- SX_REC_STOP_SPAN: Stop after the specified recording time (when the clock source is the internal clock). Stop after the specified number of points have been recorded (when the clock source is an external clock).
- SX_REC_STOP_ALARM: Stop when an alarm occurs (when the recording start condition is set to SX_REC_START_ALARM). Stop when an alarm is released (when the recording start condition is not SX_REC_START_ALARM).
- SX_REC_STOP_TRIG_RISE: Stop on the rising edge of an external trigger signal.
- SX_REC_STOP_TRIG_FALL: Stop on the falling edge of an external trigger signal.

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecStopCond(SX_HNDL_GROUP hGrp, int *stopCond);

ERR GetRecStopCond(HNDL hGrp, ref REC_STOP recCond);

Description

Queries the recording stop condition of the unit group specified by hGrp.

Parameters

hGrp A unit group handle

stopCond Where to store the recording stop condition
- SX_REC_STOP_CONTINUOUS: Continue until a recording stop command is received.
- SX_REC_STOP_TIME: Stop at the set time. (This setting is invalid when the recording start condition is set to SX_REC_START_ALARM.)
- SX_REC_STOP_SPAN: Stop after the specified recording time (when the clock source is the internal clock). Stop after the specified number of points have been recorded (when the clock source is an external clock).
- SX_REC_STOP_ALARM: Stop when an alarm occurs (when the recording start condition is set to SX_REC_START_ALARM). Stop when an alarm is released (when the recording start condition is not SX_REC_START_ALARM).
- SX_REC_STOP_TRIG_RISE: Stop on the rising edge of an external trigger signal.
- SX_REC_STOP_TRIG_FALL: Stop on the falling edge of an external trigger signal

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

Query or Set Recording Stop Time

int SxSetRecStopTime(SX_HNDL_GROUP hGrp, char *datetime);

ERR SetRecStopTime(HNDL hGrp, string datetime);

Description

Sets the recording stop time of the unit group specified by hGrp to the date and time specified by datetime. This setting is only valid when the recording stop condition is set to SX_REC_STOP_TIME.

Parameters

hGrp	A unit group handle
datetime	Where to store the date and time text string Format: "YYYY/MM/DD-hh:mm:ss" (19 characters + 1 null character) Example: Dec. 23, 2007, 15:22:33 > "2007/12/23-15:22:33"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecStopTime(SX_HNDL_GROUP hGrp, char *datetime);

ERR GetRecStopTime(HNDL hGrp, ref string datetime);

Description

Queries the recording stop time of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle
datetime	Where to store the date and time text string (20 bytes or more) Format: "YYYY/MM/DD-hh:mm:ss" (19 characters + 1 null character) Example: Dec. 23, 2007, 15:22:33 > "2007/12/23-15:22:33"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Recording Time

int SxSetRecSpan(SX_HNDL_GROUP hGrp, double second);

ERR SetRecSpan(HNDL hGrp, double second);

Description

Sets the recording time of the unit group specified by hGrp to the length in seconds specified by the second parameter.

This setting is only valid when the clock source is the internal clock and the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters

hGrp	A unit group handle
second	The length of the recording time in seconds

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecSpan(SX_HNDL_GROUP hGrp, double *second);

ERR GetRecSpan(HNDL hGrp, ref double second);

Description

Queries the recording time (in seconds) of the unit group specified by hGrp.

This setting is only valid when the clock source is the internal clock and the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters

hGrp	A unit group handle
second	Where to store the length of the recording time (in seconds)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Number of Points to Record

int SxSetRecExtClkPoints(SX_HNDL_GROUP hGrp, int points);

ERR SetRecExtClkPoints(HNDL hGrp, int points);

Description

Sets the number of points that the unit group specified by hGrp will record.
This setting is only valid when the clock source is the external clock and the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters

hGrp	A unit group handle
points	The number of points to record (the number can be set to a value from 10 to 10,000,000, but it must not exceed the recording interval).

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecExtClkPoints(SX_HNDL_GROUP hGrp, int *points);

ERR GetRecExtClkPoints(HNDL hGrp, ref int points);

Description

Queries the number of points to record in the unit group specified by hGrp.
This setting is only valid when the clock source is the external clock and the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters

hGrp	A unit group handle
points	Where to store the number of points to record

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Recording Interval Mode

int SxSetRecIntervalMode(SX_HNDL_GROUP hGrp, int mode);

ERR SetRecIntervalMode(HNDL hGrp, BOOL mode);

Description

Sets the recording interval mode of the unit group specified by hGrp to the mode specified by the mode parameter. This setting is only valid when the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters

hGrp	A unit group handle	
mode	SX_REC_INTERVAL_OFF:	Do not record at a set interval
	SX_REC_INTERVAL_ON:	Record at a set interval

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecIntervalMode(SX_HNDL_GROUP hGrp, int *mode);

ERR GetRecIntervalMode(HNDL hGrp, ref BOOL mode);

Description

Queries the recording interval mode of the unit group specified by hGrp.
This setting is only valid when the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters

hGrp	A unit group handle	
mode	Where to store the recording interval mode setting	
	SX_REC_INTERVAL_OFF:	Do not record at a set interval
	SX_REC_INTERVAL_ON:	Record at a set interval

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Recording Interval

int SxSetRecInterval(SX_HNDL_GROUP hGrp, double second);

ERR SetRecInterval(HNDL hGrp, double second);

Description

Sets the recording interval of the unit group specified by hGrp to the length in seconds specified by the second parameter.
This setting is only valid when the clock source is the internal clock, the recording stop condition is set to SX_REC_STOP_SPAN, and the recording interval mode is set to SX_REC_INTERVAL_ON.

Parameters

hGrp	A unit group handle
second	The recording interval, in seconds. The interval can be set to the second to a value from 1 to 86,400 seconds.

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecInterval(SX_HNDL_GROUP hGrp, double *second);

ERR GetRecInterval(HNDL hGrp, ref double second);

Description

Queries the recording interval (in seconds) of the unit group specified by hGrp.
This setting is only valid when the clock source is the internal clock, the recording stop condition is set to SX_REC_STOP_SPAN, and the recording interval mode is set to SX_REC_INTERVAL_ON.

Parameters

hGrp	A unit group handle
second	Where to store the recording interval (in seconds)

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

Query or Set Recording Interval Points

int SxSetRecExtClkInterval(SX_HNDL_GROUP hGrp, int interval);

ERR SetRecExtClkInterval(HNDL hGrp, int interval);

Description

Sets the number of recording interval points of the unit group specified by hGrp to the number of sample points specified by the interval parameter.
This setting is only valid when the clock source is the external clock, the recording stop condition is set to SX_REC_STOP_SPAN, and the recording interval mode is set to SX_REC_INTERVAL_ON.

Parameters

hGrp	A unit group handle
interval	The number of recording interval points (can be set to a value from 10 to 1,000,000)

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecExtClkInterval(SX_HNDL_GROUP hGrp, int *interval);

ERR GetRecExtClkInterval(HNDL hGrp, ref int interval);

Description

Queries the number of recording interval points in the unit group specified by hGrp.
This setting is only valid when the clock source is the external clock, the recording stop condition is set to SX_REC_STOP_SPAN, and the recording interval mode is set to SX_REC_INTERVAL_ON.

Parameters

hGrp	A unit group handle
interval	Where to store the number of recording interval points

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

Query or Set Record Count

int SxSetRecTimes(SX_HNDL_GROUP hGrp, int recNum);

ERR SetRecTimes(HNDL hGrp, int recNum);

Description

Sets the record count of the unit group specified by hGrp to value specified by recNum. This setting is only valid when the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters

hGrp	A unit group handle
recNum	The record count (Can be set to a value from 0 to 100,000. A value of 0 indicates no limit.)

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecTimes(SX_HNDL_GROUP hGrp, int *recNum);

ERR GetRecTimes(HNDL hGrp, ref int recNum);

Description

Queries the record count of the unit group specified by hGrp. This setting is only valid when the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters

hGrp	A unit group handle
recNum	Where to store the record count (0 indicates no limit)

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

Query or Set Recording Destination Folder (on the PC)

int SxSetRecFileFolder(SX_HNDL_GROUP hGrp, char *path);

ERR GetRecTimes(HNDL hGrp, ref int recNum);

Description

Sets the PC auto recording destination folder of the unit group specified by hGrp to the path specified by the path parameter. The maximum number of characters for the path parameter is 510.
This setting is valid when the auto recording destination is set to SX_REC_DEST_PC or SX_REC_DEST_PCUNIT.

Parameters

hGrp	A unit group handle
path	The auto recording destination folder on the PC (510 characters or less) Example: "C:\SL1000"

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecFileFolder(SX_HNDL_GROUP hGrp, char *path);

ERR GetRecFileFolder(HNDL hGrp, ref string path);

Description

Queries the PC auto recording destination folder of the unit group specified by hGrp. This setting is valid when the auto recording destination is set to SX_REC_DEST_PC or SX_REC_DEST_PCUNIT.

Parameters

hGrp	A unit group handle
path	Where to store the PC auto recording destination folder (requires 512 or more bytes of memory)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Auto Naming

int SxSetRecFileAutoNaming(SX_HNDL_GROUP hGrp, int type);

ERR SetRecFileAutoNaming(HNDL hGrp, REC_AUTONAME type);

Description

Sets the auto naming of the unit group specified by hGrp to the condition specified by the type parameter.

Parameters

hGrp	A unit group handle	
type	SX_REC_AUTONAME_DATE:	Date and time
	SX_REC_AUTONAME_NUM:	Sequential numbering

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecFileAutoNaming(SX_HNDL_GROUP hGrp, int *type);

ERR GetRecFileAutoNaming(HNDL hGrp, ref REC_AUTONAME type);

Description

Queries the auto naming setting of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle	
type	Where to store the auto naming setting	
	SX_REC_AUTONAME_DATE:	Date and time
	SX_REC_AUTONAME_NUM:	Sequential numbering

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set File Name

int SxSetRecFileName(SX_HNDL_GROUP hGrp, char *fileName);

ERR SetRecFileName(HNDL hGrp, string fileName);

Description

Sets the auto recording file name of the unit group specified by hGrp to the string specified by fileName. The maximum number of characters for the fileName parameter (for the PC) is 255, not including the null character.

This setting is valid when auto naming is set to SX_REC_AUTONAME_NUM.

Parameters

hGrp	A unit group handle
fileName	The file name (255 characters or less)
	Example: "TEST"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecFileName(SX_HNDL_GROUP hGrp, char *fileName);

ERR GetRecFileName(HNDL hGrp, ref string fileName);

Description

Queries the auto recording file name of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle
fileName	Where to store the file name (requires 256 or more bytes of memory)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set File Order

int SxSetRecFileOrder(SX_HNDL_GROUP hGrp, int fileOrder);

ERR SetRecFileOrder(HNDL hGrp, REC_FILEORDER fileOrder);

Description

Sets the file order of the unit group specified by hGrp to the condition specified by fileOrder.

Parameters

hGrp	A unit group handle	
fileOrder	SX_REC_FILEORDER_SEQUENTIAL:	Sequential
	SX_REC_FILEORDER_CYCLIC:	Cyclic

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecFileOrder(SX_HNDL_GROUP hGrp, int *fileOrder);

ERR GetRecFileOrder(HNDL hGrp, ref REC_FILEORDER fileOrder);

Description

Queries the file order of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle	
fileOrder	Where to store the file order	
	SX_REC_FILEORDER_SEQUENTIAL:	Sequential
	SX_REC_FILEORDER_CYCLIC:	Cyclic

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set File Count Limit

int SxSetRecCyclicFiles(SX_HNDL_GROUP hGrp, int fileNum);

ERR SetRecCyclicFiles(HNDL hGrp, int fileNum);

Description

Sets the file count limit of the unit group specified by hGrp to the value specified by fileNum. This setting is valid when the file order is set to SX_REC_FILEORDER_CYCLIC.

Parameters

hGrp	A unit group handle
fileNum	The file count limit (1 to 1000)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecCyclicFiles(SX_HNDL_GROUP hGrp, int *fileNum);

ERR GetRecCyclicFiles(HNDL hGrp, ref int fileNum);

Description

Queries the file count limit of the unit group specified by hGrp.
This setting is valid when the file order is set to SX_REC_FILEORDER_CYCLIC.

Parameters

hGrp	A unit group handle
fileNum	Where to store the file count limit

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Comment

int SxSetRecFileComment(SX_HNDL_GROUP hGrp, char *comment);

ERR SetRecFileComment(HNDL hGrp, string comment);

Description

Sets the comment to be saved in the auto recording file of the unit group specified by hGrp to the string specified by the comment parameter. The maximum number of characters for the comment parameter is 250, not including the null character.

Parameters

hGrp	A unit group handle
comment	The comment (250 characters or less)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetRecFileComment(SX_HNDL_GROUP hGrp, char *comment);

ERR GetRecFileComment(HNDL hGrp, ref string comment);

Description

Queries the comment to be saved in the auto recording file of the unit group specified by hGrp.

Parameters

hGrp	A unit group handle
comment	Where to store the comment (requires 251 or more bytes of memory)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Information About the File Being Recorded (on the PC)

int SxGetRecCurrentFileInfo(SX_HNDL_UNIT hUnit, char *fileName, UINT *fileSize);

ERR GetRecCurrentFileInfo(HNDL hUnit, ref string filename, ref uint filesize);

Description

Retrieves the information of the file that contains the data from the unit specified by hUnit that is being automatically recorded (saved) on the PC .

Parameters

hUnit	A unit handle
fileName	Where to store the file path (requires 512 or more bytes of memory)
fileSize	Where to store the file size (in bytes)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Information about Files That Have Been Recorded (on the PC)

int SxGetRecLastFileInfo(SX_HNDL_UNIT hUnit, char *fileName, UINT *fileSize);

ERR GetRecLastFileInfo(HNDL hUnit, ref string filename, ref uint filesize);

Description

Retrieves the file information of the file that contains the data from the unit specified by hUnit that was automatically recorded and closed on the PC last.

Parameters

hUnit	A unit handle
fileName	Where to store the file path (requires 512 or more bytes of memory)
fileSize	Where to store the file size (in bytes)

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

3.12 Measurement Controls

Start Measurement

int SxAcqStart(SX_HNDL_GROUP hGrp);

ERR AcqStart(HNDL hGrp);

Description

Causes the units in the unit group specified by hGrp to start measuring.

Parameters

hGrp A unit group handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Stop Measurement

int SxAcqStop(SX_HNDL_GROUP hGrp);

ERR AcqStop(HNDL hGrp);

Description

Causes the units in the unit group specified by hGrp to stop measuring.
If you execute this function during auto recording, the auto recording will also stop.

Parameters

hGrp A unit group handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Execute Latch (in Free Run Mode)

int SxAcqLatch(SX_HNDL_GROUP hGrp, int *point);

int SxAcqLatchL(SX_HNDL_GROUP hGrp, __int64 *point);

int SxAcqLatchD(SX_HNDL_GROUP hGrp, double *point);

ERR AcqLatch(HNDL hGrp, ref long point);

ERR AcqLatch(HNDL hGrp);

Description

Stores to num the number of sample points that have been sampled since measurement began by the units that belong to the unit group specified by hGrp. The number of sample points retrieved is the number of sample points sampled by measuring group 1.
This function is only valid in Free Run mode.
To retrieve the waveform data between the previous latch and the current latch, set the AcqNo parameter of SxGetAcqData() or SxGetAcqDataEx() to 0.
If the number of sample points exceeds 2,147,483,647, use SxAcqLatchL() or SxAcqLatchD(). SxAcqLatchL() cannot be used in VB6.

Parameters

hGrp A unit group handle

point Where to store the number of sample points that have been sampled by measuring group 1 since the start of measurement at the time the function is executed.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Confirm Measure/Save Operation

int SxIsRun(SX_HNDL_GROUP hGrp, BYTE* status);

ERR IsRun(HNDL hGrp, ref int status);

Description

Determines whether the units that belong to the group specified by hGrp are currently measuring, saving, or waiting for a trigger.

Parameters

hGrp	A unit group handle
status	Where to store the measurement condition b0: Measuring b1: Saving b2: Waiting for a trigger

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Execute Manual Trigger

int SxExecManualTrig(SX_HNDL_GROUP | SX_HNDL_UNIT hAny);

ERR ExecManualTrig(HNDL hAny);

Description

Executes a manual trigger that affects the units that belong to the object specified by hAny.

Parameters

hAny	Any kind of handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.13 Auto Recording Controls

Start Recording

int SxRecStart(SX_HNDL_GROUP hGrp);

ERR RecStart(HNDL hGrp);

Description

Causes the units in the unit group specified by hGrp to start auto recording.
If you execute this function when measurement has not started, measurement will start.
Use this function after setting the auto recording condition with the set auto recording condition function.

Parameters

hGrp A unit group handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Stop Recording

int SxRecStop(SX_HNDL_GROUP hGrp);

ERR RecStop(HNDL hGrp);

Description

Causes the units in the unit group specified by hGrp to stop auto recording.
Measurement will continue even after this function is executed.

Parameters

hGrp A unit group handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Divide File

int SxRecDevide(SX_HNDL_GROUP hGrp);

ERR RecDevide(HNDL hGrp);

Description

If the units that belong to the unit group specified by hGrp are auto recording in Free Run mode, this function closes the currently recorded file and continues recording on a new file.

Parameters

hGrp A unit group handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.14 Acquisition and Deletion of Measured Data

Get Acquisition Data Information (Constants for Converting Physical Values)

int SxGetChannelInfo(SX_HNDL_CH hCh, SX_INFO_CH *inf);

ERR GetChannelInfo(HNDL hCh, ref INFO_CH inf);

Description

Retrieves the acquisition data information for converting the measured data from the channel specified by hCh to physical values.

Parameters

hCh	A channel handle
inf	Where to store the retrieved data information

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Latch Interval Sample Points

int SxGetLatchLength(SX_HNDL_MEASGRP | SX_HNDL_GROUP hAny, int *len);

ERR GetLatchLength(HNDL hAny, ref long len);

Description

Returns the latch interval sample points.

If hAny is a measuring group, the function returns the number of sample points in the measuring group. If hAny is a unit group, the function returns the number of sample points in measuring group 1 in the unit group. This function is valid in Free Run mode.

Parameters

hAny	Any kind of handle
len	Where to store the number latch interval sample points

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Latest Acquisition Number

int SxGetLatestAcqNo(SX_HNDL_GROUP hGrp, int *acqNo);

int SxGetLatestAcqNoL(SX_HNDL_GROUP hGrp, __int64 *acqNo);

int SxGetLatestAcqNoD(SX_HNDL_GROUP hGrp, double *acqNo);

ERR GetLatestAcqNo(HNDL hAny, ref long acqNo);

Description

Returns the most recently retrieved acquisition number.

If the number of acquisitions exceeds 2,147,483,647, use SxGetLatestAcqNoL() or SxGetLatestAcqNoD(). SxGetLatestAcqNoL() cannot be used in VB6.

Parameters

hGrp	A unit group handle
acqNo	Where to store the latest acquisition number

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Waveform Data

int SxGetAcqData(SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int acqNo, void* buf, int blen, int *rlen);

int SxGetAcqDataL(SX_HNDL_CH | SX_HNDL_MEASGRP hAny, __int64 acqNo, void* buf, int blen, int *rlen);

int SxGetAcqDataD(SX_HNDL_CH | SX_HNDL_MEASGRP hAny, double acqNo, void* buf, int blen, int *rlen);

ERR GetAcqData(HNDL hAny, long acqNo, ref any[] buf, int blen, ref int rlen);

ERR GetAcqData(HNDL hAny, long acqNo, ref any[] buf, ref int rlen);

Description

Stores the waveform data with the acquisition number specified by acqNo to the memory location specified by buf.

The values stored in buf are A/D converted values. To convert the data into physical values, the vResolution and vOffset acquisition data information values are necessary.

If the waveform data from multiple channels is stored, the data will be stored in block format starting with the lowest numbered channel's data.

Use blen to specify the size in bytes of buf.

The actual stored byte size will be stored in rlen.

If hAny is a channel handle, only the waveform data for that channel will be stored.

If hAny is a measuring group, the waveform data of all of the channels in the measuring group will be stored.

If the number of acquisitions exceeds 2,147,483,647, use SxGetAcqDataL() or SxGetAcqDataD(). SxGetAcqDataL() cannot be used in VB6.

Parameters

hAny	Any kind of handle	
acqNo	An acquisition number	
	Positive number:	An absolute acquisition number
	Negative number:	A relative number with –1 being the most recent acquisition.
	Zero:	All of the history (in Triggered mode) The latch interval (in Free Run mode)
buf	Where to store the waveform data	
blen	The size of the memory area where the waveform data is stored (in bytes)	
rlen	Where to store the size of the stored waveform data	

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Waveform Data Segment

int SxGetAcqDataEx(SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int acqNo, int start, int count, void* buf, int blen, int *rlen);

int SxGetAcqDataExL(SX_HNDL_CH | SX_HNDL_MEASGRP hAny, __int64 acqNo, __int64 start, int count, void* buf, int blen, int *rlen);

int SxGetAcqDataExD(SX_HNDL_CH | SX_HNDL_MEASGRP hAny, double acqNo, double start, int count, void* buf, int blen, int *rlen);

ERR GetAcqDataEx(HNDL hAny, long acqNo, long start, int count, ref any[] buf, int blen, ref int rlen);

ERR GetAcqDataEx(HNDL hAny, long acqNo, long start, int count, ref any[] buf, ref int rlen);

Description

Stores the waveform data with the acquisition number specified by acqNo to the memory location specified by buf. The values stored in buf are A/D converted values. To convert the data into physical values, the vResolution and vOffset acquisition data information values are necessary.

If the waveform data from multiple channels is stored, the data will be stored in block format starting with the lowest numbered channel's data.

Specify the beginning of the waveform data segment that you want to retrieve using the start parameter. Specify the value using the sampling units of measuring group 1.

Specify the amount waveform data that you want to retrieve using the count parameter. Specify the value using the sampling units of measuring group 1. If you specify 0 for count, all of the data from the beginning of the segment to the end of the waveform data will be retrieved.

Use blen to specify the size in bytes of buf. The actual stored byte size will be stored in rlen.

If hAny is a channel handle, only the waveform data for that channel will be stored.

If hAny is a measuring group, the waveform data of all of the channels in the measuring group will be stored.

If the number of acquisitions and measurement points exceeds 2,147,483,647, use SxGetAcqDataExL() or SxGetAcqDataExD(). SxGetAcqDataExL() cannot be used in VB6.

Parameters

hAny	Any kind of handle	
acqNo	An acquisition number	
	Positive number:	An absolute acquisition number
	Negative number:	A relative number with –1 being the most recent acquisition.
	Zero:	All of the history (in Triggered mode) The latch interval (in Free Run mode)
start	The start of the data segment to be retrieved (in the sampling units of measuring group 1, starting from 0)	
count	The length of the data segment to be retrieved (in the sampling units of measuring group 1, starting from 0)	
buf	Where to store the waveform data	
blen	The size of the memory area where the waveform data is stored (in bytes)	
rlen	Where to store the size of the stored waveform data	

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Data Acquisition Time

int SxGetAcqTime(SX_HNDL_MEASGRP | SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny, int acqNo, char *datetime);

int SxGetAcqTimeL(SX_HNDL_MEASGRP | SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny, __int64 acqNo, char *datetime);

int SxGetAcqTimeD(SX_HNDL_MEASGRP | SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny, double acqNo, char *datetime);

ERR GetAcqTime(HNDL hAny, long acqNo, ref string datetime);

Description

Retrieves the trigger time of the waveform data with the acquisition number specified by acqNo in the unit that belongs to the object specified by hAny (if hAny is a unit group or measuring group, the master unit is used). The acqNo specification is invalid in Free Run mode, and the time when measurement started is acquired.

Allocate at least 27 bytes of memory area for datetime.

If the number of acquisitions exceeds 2,147,483,647, use SxGetAcqTimeL() or SxGetAcqTimeD(). SxGetAcqTimeL() cannot be used in VB6.

Parameters

hAny	Any kind of handle	
acqNo	An acquisition number (this parameter is invalid in Free Run mode)	
	Positive number:	An absolute acquisition number
	Negative number:	A relative number with –1 being the most recent acquisition.
datetime	Where to store the date and time text string	
	Format:	"YYYY/MM/DD-hh:mm:ss.uuuuuu" (26 characters + 1 null character)
	Example:	Dec. 23, 2007, 15:22:33.123456 > "2007/12/23-15:22:33.123456"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Phase Difference between Measuring Groups

int SxGetAcqDelay(SX_HNDL_MEASGRP hMgrp, int acqNo, int *delay);
int SxGetAcqDelayL(SX_HNDL_MEASGRP hMgrp, __int64 acqNo, int *delay);
int SxGetAcqDelayD(SX_HNDL_MEASGRP hMgrp, double acqNo, int *delay);
ERR GetAcqDelay(HNDL hAny, long acqNo, ref int delay);

Description

Retrieves the difference in measurement points between (1) the first sample point in the waveform data with the acquisition number specified by acqNo in the measurement group specified by hMgrp and (2) the first sample point in measuring group 1. One measurement point is equal to one measuring group 1 sample interval. The function returns zero if there is no difference.
If the number of acquisitions exceeds 2,147,483,647, use SxGetAcqDelayL() or SxGetAcqDelayD(). SxGetAcqDelayL() cannot be used in VB6.

Parameters

hMgrp	A measuring group handle	
acqNo	An acquisition number (this parameter is invalid in Free Run mode)	
	Positive number:	An absolute acquisition number
	Negative number:	A relative number with –1 being the most recent acquisition.
delay	Where to store the difference between the sampling start points of measuring group 1 and the specified waveform data (from 0).	

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

Create WDF File

int SxSaveAcqData(SX_HNDL_UNIT | SX_HNDL_CH hAny, int acqNo, char *fileName);
int SxSaveAcqDataL(SX_HNDL_UNIT | SX_HNDL_CH hAny, __int64 acqNo, char *fileName);
int SxSaveAcqDataD(SX_HNDL_UNIT | SX_HNDL_CH hAny, double acqNo, char *fileName);
ERR SaveAcqData(HNDL hAny, long acqNo, string fileName);

Description

Creates a WDF file with the file name specified by fileName from the waveform data that has the acquisition number specified by acqNo.
If hAny is a unit, the WDF file will contain all active channels. If hAny is a channel, the WDF file will only contain the waveform data for that channel.
If acqNo is positive, the file will be created using the waveform data that has the specified acquisition number. If acqNo is negative, the file will be created using the relative waveform data that has been acquired in the order specified by acqNo, with –1 as the most recent data, –2 as the data before that, –3 as the data before that, etc. If acqNo is zero, all history data will be saved.
The extension is .wdf. The extension is added automatically, so it does not need to be included in fileName. This function is valid in Triggered mode.
If the number of acquisitions exceeds 2,147,483,647, use SxSaveAcqDataL() or SxSaveAcqDatayD(). SxSaveAcqDataL() cannot be used in VB6.

Parameters

hAny	Any kind of handle	
acqNo	An acquisition number	
	Positive number:	The specified number
	Negative number:	A relative number with –1 being the most recent acquisition.
	Zero:	All of the history (in Triggered mode)
fileName	The file name	

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

Create WDF File (from Waveform Data Segment)

int SxSaveAcqDataEx(SX_HNDL_UNIT | SX_HNDL_CH hAny, int acqNo, int start, int count, char *fileName);

int SxSaveAcqDataExL(SX_HNDL_UNIT | SX_HNDL_CH hAny, __int64 acqNo, int start, int count, char *fileName);

int SxSaveAcqDataExD(SX_HNDL_UNIT | SX_HNDL_CH hAny, double acqNo, int start, int count, char *fileName);

ERR SaveAcqDataEx(HNDL hAny, long acqNo, long start, int count, string filename);

Description

Creates a WDF file with the file name specified by fileName from the waveform data that has the acquisition number specified by acqNo.

Specify the beginning of the waveform data segment that you want to save using the start parameter. Specify the value using the sampling units of measuring group 1.

Specify the amount waveform data that you want to save using the count parameter. Specify the value using the sampling units of measuring group 1. If you specify 0 for count, all of the data from the beginning of the segment to the end of the waveform data will be saved.

If hAny is a unit, the WDF file will contain all active channels. If hAny is a channel, the WDF file will only contain the waveform data for that channel.

If acqNo is positive, the file will be created using the waveform data that has the specified acquisition number. If acqNo is negative, the file will be created using the relative waveform data that has been acquired in the order specified by acqNo, with –1 as the most recent data, –2 as the data before that, –3 as the data before that, etc. If acqNo is zero, all history data will be saved.

The extension is .wdf. The extension is added automatically, so it does not need to be included in fileName. This function is valid in Triggered mode.

If the number of acquisitions exceeds 2,147,483,647, use SxSaveAcqDataExL() or SxSaveAcqDatayExD(). SxSaveAcqDataExL() cannot be used in VB6.

Parameters

hAny	Any kind of handle	
acqNo	An acquisition number	
	Positive number:	The specified number
	Negative number:	A relative number with –1 being the most recent acquisition.
	Zero:	All of the history (in Triggered mode)
fileName	The file name	

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Instantaneous Values

int SxGetCurrentData(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP|SX_HNDL_CHGRP hAny, void* buf, int blen, int *rlen);

ERR GetCurrentData(HNDL hAny, ref any[] buf, int blen, ref int rlen);

ERR GetCurrentData(HNDL hAny, ref any[] buf, ref int rlen);

ERR GetCurrentData(HNDL hAny, ref any buf);

Description

Stores instantaneous values (the most recently measured values) in the memory area specified by buf.

The values stored in buf are A/D converted values. To convert the data into physical values, the vResolution and vOffset acquisition data information values are necessary.

If the instantaneous values from multiple channels is stored, the data will be stored in block format starting with the lowest numbered channel's data.

Use blen to specify the size in bytes of buf.

The actual stored byte size will be stored in rlen.

If hAny is a channel handle, only the instantaneous values for that channel will be stored.

If hAny is a module handle, all of the instantaneous values of the channels in that module will be stored.

If hAny is a unit handle, all of the instantaneous values of the channels in that unit will be stored.

If hAny is a unit group handle, all of the instantaneous values of the channels in that unit group will be stored.

Parameters

hAny	Any kind of handle
buf	Where to store the instantaneous values
blen	The size of the memory area where the instantaneous values are stored (in bytes)
rlen	Where to store the size of the stored instantaneous values

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Delete Waveform Data

int SxClearAcqData(SX_HNDL_GROUP | SX_HNDL_UNIT hAny);

ERR SaveSetup(HNDL hUnit, string fileName);

Description

Deletes all of the waveform data (clears the entire history) of the units that belong to the object specified by hAny.

Parameters

hAny	Any kind of handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.15 Setup Data Access

Save Setup Data

int SxSaveSetup(SX_HNDL_UNIT hUnit, char *fileName);

ERR SaveSetup(HNDL hUnit, string fileName);

Description

Saves the setup data of the unit specified by hUnit to the file (on the PC) specified by fileName.

Parameters

hUnit	A unit handle
fileName	A file name

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Load Setup Data

int SxLoadSetup(SX_HNDL_UNIT hUnit, char *fileName);

ERR LoadSetup(HNDL hUnit, string fileName);

Description

Loads the setup data stored to the file (on the PC) specified by fileName to the unit specified by hUnit.

Parameters

hUnit	A unit handle
fileName	A file name

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Initialize Setup Data

int SxInitSetup(SX_HNDL_GROUP | SX_HNDL_UNIT hAny);

ERR InitSetup(HNDL hAny);

Description

Returns the setup data on the units specified by hAny to the factory default settings of those units.

If hAny is a unit group, the function initializes the setup data of all of the units in that group.

Parameters

hAny	Any kind of handle

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.16 System-Related Functions

Execute Calibration

int SxExecCal(SX_HNDL_UNIT hUnit, int *result);

ERR ExecCal(HNDL hUnit, ref int result);

Description

Calibrates the unit specified by hUnit.
This function returns after calibration is complete.

Parameters

hUnit	A unit handle
result	Where to store the results of calibration Zero: Succeeded Nonzero: Failed

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Execute Self Test

int SxExecSelftest(SX_HNDL_UNIT hUnit, int *result);

ERR ExecSelftest(HNDL hUnit, ref int result);

Description

Executes a self test on the unit specified by hUnit.
This function returns after self testing is complete.

Parameters

hUnit	A unit handle
result	Where to store the results of the self test The results of the test are indicated using the logical sum of these constants: SX_SELFTEST_ACQMEM: Waveform memory error SX_SELFTEST_SYSMEM: System error SX_SELFTEST_BACKUPMEM: Backup memory error SX_SELFTEST_HDD: Internal hard disk error If there are no errors, 0 will be stored.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Unit CPU Temperature

int SxGetCpuTemperature(SX_HNDL_UNIT hUnit, int *temperature);

ERR GetCpuTemperature(HNDL hUnit, ref int temperature);

Description

Retrieves the CPU temperature of the unit specified by hUnit.

Parameters

hUnit	A unit handle
temperature	Where to store the temperature (in °C)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Unit Clock

int SxSetSystemClock(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *datetime);

ERR SetSystemClock(HNDL hAny, string datetime);

Description

Sets the system clocks of the units that belong to the object specified by hAny.

Parameters

hAny	Any kind of handle
datetime	A date and time text string (the time can be specified to the microsecond) Format: "YYYY/MM/DD-hh:mm:ss.uuuuuu" (26 characters + 1 null character) Example: Dec. 23, 2007, 15:22:33.123456 > "2007/12/23-15:22:33.123456"

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetSystemClock(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *datetime);

ERR GetSystemClock(HNDL hAny, ref string datetime);

Description

Queries the system clock of the unit that belongs to the object specified by hAny.
Allocate at least 27 bytes of memory area for datetime.
If hAny is a unit group, the function queries the master unit's system clock.

Parameters

hAny	Any kind of handle
datetime	Where to store the date and time text string (the time can be specified to the microsecond) Format: "YYYY/MM/DD-hh:mm:ss.uuuuuu" (26 characters + 1 null character) Example: "2007/12/23-15:22:33.123456" > Dec. 23, 2007, 15:22:33.123456

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

Query or Set DHCP

int SxSetEthernetDHCP(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int mode);

ERR SetEthernetDHCP(HNDL hAny, int mode);

Description

Sets the DHCP client of the units that belong to the object specified by hAny.

Parameters

hAny Any kind of handle

mode SX_DHCP_OFF: Do not use DHCP
SX_DHCP_ON: Use DHCP

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetEthernetDHCP(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int *mode);

ERR GetEthernetDHCP(HNDL hAny, ref int mode);

Description

Queries the DHCP client setting of the unit specified by hAny.
If hAny is a unit group, the function queries the master unit's settings.

Parameters

hAny Any kind of handle

mode Where to store the mode
SX_DHCP_OFF: Do not use DHCP
SX_DHCP_ON: Use DHCP

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set IP address

int SxSetEthernetIP(SX_HNDL_UNIT hUnit, char *adrs);

ERR SetEthernetIP(HNDL hUnit, string adrs);

Description

Sets the Ethernet IP address of the unit specified by hUnit.

Parameters

hUnit A unit handle

adrs An IP address
Example: "192.168.21.3"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetEthernetIP(SX_HNDL_UNIT hUnit, char *adrs);

ERR GetEthernetIP(HNDL hUnit, ref string adrs);

Description

Queries the Ethernet IP address of the unit specified by hAny.

Parameters

hUnit A unit handle

adrs Where to store the IP address

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Subnet Mask

int SxSetEthernetNetMask(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *adrs);

ERR SetEthernetNetMask(HNDL hAny, string adrs);

Description

Sets the subnet of the units that belong to the object specified by hAny.

Parameters

hAny Any kind of handle

adrs A subnet mask
Example: "255.255.255.0"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetEthernetNetMask(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *adrs);

ERR GetEthernetNetMask(HNDL hAny, ref string adrs);

Description

Queries the subnet mask of the unit specified by hAny.
If hAny is a unit group, the function queries the master unit's settings.

Parameters

hAny Any kind of handle

adrs Where to store the subnet mask

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Default Gateway

int SxSetEthernetGateway(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *adrs);

ERR SetEthernetGateway(HNDL hAny, string adrs);

Description

Sets the default gateway address of the units that belong to the object specified by hAny.

Parameters

hAny Any kind of handle

adrs A default gateway address
Example: "192.168.21.254"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetEthernetGateway(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *adrs);

ERR GetEthernetGateway(HNDL hAny, ref string adrs);

Description

Queries the default gateway address of the unit specified by hAny.
If hAny is a unit group, the function queries the master unit's settings.

Parameters

hAny Any kind of handle

adrs Where to store the default gateway address

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set SNTP

int SxSetSNTP(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int mode, char *serverAddress, char *timeZone, int interval);

ERR SetSNTP(HNDL hAny, int mode, string serverAddress, string timeZone, int interval);

Description

Sets the SNTP client of the units that belong to the object specified by hAny.
If you execute this function with mode=1, the time will be set when the function is executed.
After that, the time will be set at the interval specified in hours by the interval parameter.

Parameters

hAny	Any kind of handle
mode	SX_SNTP_DISABLE: Do not use SNTP SX_SNTP_ENABLE: Use SNTP
serverAddress	The SNTP server address Example: "192.168.21.1"
timeZone	The time difference from UTC. Format: "hh:mm" or "–hh:mm" Example: In Japan, the time difference would be "09:00"
interval	The interval at which to set the time (specified in hours) You can set the interval to 0, 1, 2, 4, 6, 8, 12, or 24 hours. If you specify 0, the time will not be set at regular intervals.

Return Value

Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetSNTP(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int *mode, char *serverAddress, char *timeZone, int *interval);

ERR GetSNTP(HNDL hAny, ref int mode, ref string serverAddress, ref string timeZone, ref int interval);

Description

Queries the SNTP client setting of the unit that belongs to the object specified by hAny.
If hAny is a unit group, the function queries the master unit's settings.
Allocate at least 16 bytes of memory area for serverAddress.

Parameters

hAny	Any kind of handle
mode	Where to store the mode SX_SNTP_DISABLE: Do not use SNTP SX_SNTP_ENABLE: Use SNTP
serverAddress	Where to store the SNTP server address
timeZone	The time difference from UTC. Format: "hh:mm" or "–hh:mm" Example: In Japan, the time would be "09:00"
interval	Where to store the interval at which to set the time (specified in hours) Zero indicates that the time will not be set at regular intervals.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Key Lock

int SxSetKeyLock(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int mode);

ERR SetKeyLock(HNDL hAny, BOOL mode);

Description

Sets the key lock setting of the units that belong to the object specified by hAny.

Parameters

hAny	Any kind of handle
mode	SX_KEYLOCK_OFF: Do not lock keys SX_KEYLOCK_ON: Lock keys

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetKeyLock(SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int mode);

ERR GetKeyLock(HNDL hAny, ref BOOL mode);

Description

Queries the key lock setting of the unit specified by hAny.
If hAny is a unit group, the function queries the master unit's settings.

Parameters

hAny	Any kind of handle
mode	Where to store the mode SX_KEYLOCK_OFF: Do not lock keys SX_KEYLOCK_ON: Lock keys

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Error Code

int SxGetError(SX_HNDL_UNIT hUnit, int *error);

ERR GetError(HNDL hAny, ref int error);

Description

Retrieves an error code from the error cue of the unit specified by hUnit. If hUnit is a unit group, the error code is retrieved from the master unit.
If the error cue is empty, the function returns zero.

Parameters

hUnit	A unit handle
error	Where to store the error code Zero: The error cue is empty Nonzero: An error code

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.17 Internal Media Operations

Query Drive Count

int SxFileGetDriveNum(SX_HNDL_UNIT hUnit, int *driveNum);

ERR FileGetDriveNum(HNDL hUnit, ref int driveNum);

Description

Retrieves the number of internal media drives of the unit specified by hUnit.

Parameters

hUnit	A unit handle
driveNum	Where to store the number of drives

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query Drive Information

int SxFileGetDriveInfo(SX_HNDL_UNIT hUnit, int index, char *driveName, int *totalSize, int *freeSize);

ERR FileGetDriveInfo(HNDL hUnit, int index, ref string driveName, ref int totalSize, ref int freeSize);

Description

Retrieves the drive information of the drive with the specified index in the unit specified by hUnit.

Allocate at least 8 bytes of memory area for driveName.

Parameters

hUnit	A unit handle
index	A drive number (starting with 0)
driveName	Where to store the drive name Example: "HD-0"
totalSize	Where to store the total size (in kilobytes)
freeSize	Where to store the available memory (in kilobytes)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Current Drive

int SxFileSetCurrentDrive(SX_HNDL_UNIT hUnit, char *driveName);

ERR FileSetCurrentDrive(HNDL hUnit, string driveName);

Description

Sets the current drive (the target drive) of the unit specified by hUnit to the drive specified by driveName.

Parameters

hUnit	A unit handle
driveName	A drive name Example: "HD-0"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxFileGetCurrentDrive(SX_HNDL_UNIT hUnit, char *driveName);

ERR FileGetCurrentDrive(HNDL hUnit, ref string driveName);

Description

Retrieves the current drive (the target drive) of the unit specified by hUnit. Allocate at least 8 bytes of memory area for driveName.

Parameters

hUnit	A unit handle
driveName	Where to store the drive name (requires 8 or more bytes of memory)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Current Directory

int SxFileChDir(SX_HNDL_UNIT hUnit, char *pathName);

ERR FileChDir(HNDL hUnit, string pathName);

Description

Set the current directory (the target directory) of the unit specified by hUnit to the directory specified by pathName.

Parameters

hUnit	A unit handle
pathName	A directory path Example: "/abcdefg/efghi" > An absolute path specification "jklmn" > A relative path specification ".." > To a new directory "/" > To the root directory

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxFileCwDir(SX_HNDL_UNIT hUnit, char *pathName);

ERR FileCwDir(HNDL hUnit, ref string pathName);

Description

Retrieves the absolute path name of the current directory (the target directory) of the unit specified by hUnit. Allocate at least 256 bytes of memory area for pathName.

Parameters

hUnit	A unit handle
pathName	Where to store the path (requires 256 or more bytes of memory) Example: "/abcdefg/efghi"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Create Subdirectory

int SxFileMkDir(SX_HNDL_UNIT hUnit, char *dirName);

ERR FileMkDir(HNDL hUnit, string dirName);

Description

Creates a subdirectory with the name specified by dirName in the current directory of the unit specified by hUnit.

Parameters

hUnit	A unit handle
dirName	The name of the subdirectory Example: "jklmn"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Delete Subdirectory

int SxFileRmDir(SX_HNDL_UNIT hUnit, char *dirName);

ERR FileRmDir(HNDL hUnit, string dirName);

Description

Deletes the subdirectory with the name specified by dirName in the current directory of the unit specified by hUnit. All of the files in the directory are also deleted.

Parameters

hUnit	A unit handle
dirName	A subdirectory name Example: "jklmn"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get File Number

int SxFileGetFileNum(SX_HNDL_UNIT hUnit, int *fileNum);

ERR FileGetFileNum(HNDL hUnit, ref int fileNum);

Description

Retrieves the number of files (including subdirectories) in the current directory of the unit specified by hUnit. Use this function before getting file information with SxFileGetFileInfo().

Parameters

hUnit	A unit handle
dirName	A subdirectory name Example: "jklmn"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get File Information

int SxFileGetFileInfo(SX_HNDL_UNIT hUnit, int index, char *fileName, int *attr, int *size, char *datetime);

ERR FileGetFileInfo(HNDL hUnit, int index, ref string fileName, ref int attr, ref int size, ref string datetime);

Description

Retrieves the information of the file specified by index in the current directory of the unit specified by hUnit. Allocate at least 256 bytes of memory area for fileName.
Allocate at least 17 bytes of memory area for datetime.
Before using this function, execute SxFileGetFileNum() after setting the appropriate current directory.

Parameters

hUnit	A unit handle
index	A file number (starting with 0)
fileName	Where to store the file name (requires 256 or more bytes of memory) Example: "stuvw.xyz"
attr	Where to store the file attributes The attributes are expressed using the logical sum of the following values. SX_ATTR_SUBDIR: Subdirectory (Currently, only the Subdirectory attribute listed above is defined.)
size	Where to store the file size (in bytes)
datetime	Where to store the file date and time (requires 17 or more bytes of memory) Format: "YYYY/MM/DD-hh:mm" (16 characters + 1 null character) Example: Dec. 23, 2007, 15:22:33 > "2007/12/23-15:22:33"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Delete File

int SxFileDelete(SX_HNDL_UNIT hUnit, char *fileName);

ERR FileDelete(HNDL hUnit, string fileName);

Description

Deletes the file with the name specified by fileName in the current directory of the unit specified by hUnit. You cannot delete subdirectories with this function. To delete a subdirectories, use SxFileRmDir().

Parameters

hUnit	A unit handle
fileName	A file name Example: "stuvw.xyz"

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get File

int SxFileGet(SX_HNDL_UNIT hUnit, char *srcFileName, char *dstFileName);
int SxFileGetM(SX_HNDL_UNIT hUnit, char *srcFileName, char *buf, int bsize, int *size);
ERR FileGet(HNDL hUnit, string srcFileName, string dstFileName);
ERR FileGetM(HNDL hUnit, string srcFileName, ref any[] buf, int bsize, ref int size);
ERR FileGetM(HNDL hUnit, string srcFileName, ref any[] buf, ref int size)

Description

Retrieves the file with the name specified by srcfileName in the current directory of the unit specified by hUnit.
SxFileGet() copies the contents of the retrieved file to the local file specified by dstFileName.
SxFileGetM() copies the contents of the retrieved file to the memory address specified by buf.

Parameters

hUnit	A unit handle
srcFileName	The name of the copy source file (on the unit) Example: "stuvw.xyz"
dstFileName	The name of the copy destination file (on the PC) Example: "C:\SL1000\stuvw.xyz"
buf	Where to store the file image
bsize	The buffer size (in bytes) of where the file image will be stored.
size	Where to store the actual size of the file image (in bytes)

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Create File

int SxFilePut(SX_HNDL_UNIT hUnit, char *dstFileName, char *srcFileName);
int SxFilePutM(SX_HNDL_UNIT hUnit, char *dstFileName, char *buf, int size);
ERR FilePut(HNDL hUnit, string dstFileName, string srcFileName);
ERR FilePutM(HNDL hUnit, string dstFileName, ref any[] buf, int size);
ERR FilePutM(HNDL hUnit, string dstFileName, ref any[] buf);

Description

Creates a file with the name specified by dstfileName in the current directory of the unit specified by hUnit.
SxFilePut() copies the contents of the local file specified by srcFileName.
SxFilePutM() copies the contents of the memory address specified by buf.

Parameters

hUnit	A unit handle
dstFileName	The name of the file to be created (on the unit) Example: "stuvw.xyz"
srcFileName	The name of the copy source file (from the PC) Example: "C:\SL1000\stuvw.xyz"
buf	Where to store the file image
size	The file image size in bytes

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

3.18 Debugging

Query or Set Trace Mode

int SxTraceSetMode(int mode, const char *fileName);

ERR TraceSetMode(int mode, string fileName);

ERR TraceSetMode(int mode);

Description

Sets the trace mode to the mode specified by the mode parameter. The fileName parameter specifies the file to write to (a local PC file) if file output is set to on.
The application program can execute this function at any time.

Parameters

mode	The trace mode Specified using a logical sum. For information about the values that can be set, see section 3.19.
fileName	The file to write to when file output is set to on. If the valuel is null, the function will write to the default file name (".\SxAPILog.txt").

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Explanation

The following two methods are also available for changing the trace mode.

With environment variables

Environment variable	Value
SXAPI_TRACE_MODE	The trace mode (example: 0x1131)
SXAPI_TRACE_FILE	Trace file name (example: C:\TEMP\SxAPI.log)

With command line parameters

Format	Example
-SXAPI_TRACE_MODE = <the trace mode>	-SXAPI_TRACE_MODE=0x1131
-SXAPI_TRACE_FILE = "<the trace file name>"	-SXAPI_TRACE_MODE="C:\TEMP\SxAPI.log"

The order of preference for the settings is: those settings made by this function, then command line parameters, then environment variables.
The trace mode default value is zero (no tracing), and the default trace file name is ".\SxTraceLog.txt."

int SxTraceGetMode(int *mode, char *fileName);

ERR TraceGetMode(ref int mode, ref string fileName);

Description

Retrieves the trace mode and the trace output file name (on the PC).

Parameters

mode	Where to store the trace mode Expressed using a logical sum. For information about the values that can be set, see section 3.19.
fileName	Where to store the name of the file that is written to when file output is set to on.

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Output Trace

int SxTracePrint(char *s);

int SxTracePrintf(char *fmt, ...);

ERR TracePrint(string s);

Description

SxTracePrint() outputs the string specified by the s parameter to the trace log.
SxTracePrintf() outputs the specified string using the same format as printf().
This function can only be used with VC++.

Parameters

s	A string (ending with a null character)
fmt	A string in printf() format

Return Value

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Reset Performance Timer

unsigned long SxResetTimer();

uint ResetTimer();

Description

Executing this function resets the performance timer to 0. The function returns the timer value (in milliseconds) immediately preceding the reset.

Parameters

None

Return Value

unsigned long	The timer value immediately preceding the reset (in microseconds)

Explanation

The performance timer is reset to zero when an SxAPI function is first called (this first function is ordinarily SxIint). Afterwards, the timer continues counting until this function is executed.
The performance timer is accurate to the microsecond.

Get Performance Timer

unsigned long SxGetTimer();

uint GetTimer();

Description

This function returns the current performance timer value (in microseconds) when it is executed.

Parameters

None

Return Value

unsigned long The timer value immediately preceding the reset (in microseconds)

3.19 Definitions

Handles

```
typedef unsigned long          SX_HNDL;    // Any type of handle
#define SX_HNDL_COMM           SX_HNDL     // Communication handle
#define SX_HNDL_GROUP          SX_HNDL     // Unit group handle
#define SX_HNDL_UNIT           SX_HNDL     // Unit handle
#define SX_HNDL_MOD            SX_HNDL     // Module handle
#define SX_HNDL_CH             SX_HNDL     // Channel handle
#define SX_HNDL_MEASGRP        SX_HNDL     // A measuring group handle
```

Communication Types

```
#define SX_WIRE_USB            7           // USB (USBTMC)
#define SX_WIRE_LAN            8           // Ethernet (VXI-11)
```

Module Information Structure

```
typedef struct {
    int     slotNo;                // Slot number
    int     type;                  // 0: empty, 1: AcqModule
    int     measGroupNo;           // A measuring group number (0 to 3)
    int     chOffsetGrp;           // Internal unit group channel offset number
    int     chOffsetUnit;          // Internal unit channel offset number
    int     chNum;                 // Number of channels
    char    productCode[16];       // Module code (example: "M701250")
    char    productName[16];       // Module name (example: "HS10M12")
}SX_INFO_MOD;
```

Unit Information Structure

```
typedef struct {
    int     groupID;               // Unit group number
    int     unitID;                // Unit number
    char    address[64];           // Unit address
    int     chOffsetGrp;           // Internal unit group channel offset number
    int     modNum;                // Number of modules
    char    productCode[16];       // Product model (example: "720120")
    char    groupName[32];         // Unit name (example: "Unit1")
    char    unitName[32];          //Options (example: "128MW,HD,ETHER")
    SX_INFO_MOD moduleInfo[10];// Module information
} SX_INFO_UNIT;
```

Unit Group Information Structure

```
typedef struct {
    int     groupID;               // Unit group number
    int     errorCode;             // Unit group error code (can be opened if the code is
                                   // 0). For details, see “Unit Group Error Codes.”
    int     unitNum;               // Number of units
    char    groupName[32];         // When errorCode!=0 the value is “” (empty)
} SX_INFO_GROUP;
```

Acquisition Data Information Structure

```
typedef struct {
    INT         channel;        // Logic channel number (starting with 0)
    UCHAR       dataType;       // Raw data type
    CHAR        startBit;       // Location of the first effective bit (starting with 0) in
                                // an integer.
    CHAR        effectiveBit;   // Effective bit length of an integer (0 is valid until lsb)
    char        pad1[1]
    char        label[8]        // Label name
    double      vResolu;        // Physical value conversion gain
    double      vOffset;        // Physical value conversion offset
    UNION64     nonData;        // Non-display code (raw data or something close)
    char        unit[4];        // Unit of measurement string
    char        pad2[4];
    INT         recordLen;      // Measurement points (data points)
    INT         trigPos;        // Trigger position (Only valid in Triggered mode.
                                // The value is 0x80000000 when it is invalid.)
    char        pad3[8];
    char        pad4[8];
    char        pad5[8];
} SX_INFO_CH;
```

Event Types

```
#define SX_EV_ALA                   0x00000080  // System alarm has occurred.
#define SX_EV_ACQ_START             0x00010000  // Measurement has started.
#define SX_EV_ACQ_STOP              0x00020000  // Measurement has ended.
#define SX_EV_TRIG_START            0x00040000  // A trigger has been detected (in
                                                   Triggered mode).
#define SX_EV_TRIG_END              0x00080000  // A single triggered measurement
                                                   has ended.
#define SX_EV_REC_START             0x01000000  // Recording has started.
#define SX_EV_SAVE_SATRT            0x02000000  // The SL1000 has started saving
                                                   waveform data.
#define SX_EV_REC_END               0x04000000  // Recording has ended.
#define SX_EV_ACQ_DATA_READY        0x08000000  // The specified number of data
                                                   points have been acquired (in
                                                   Free Run mode).
#define SX_EV_SAVE_END              0x10000000  // The SL1000 has finished saving
                                                   waveform data
#define SX_EV_CHANNEL_ALARM         0x20000000  // A channel alarm has occurred
                                                   or been released
```

Error Codes

```
#define SX_ERR_OK                   0       // No error, closed properly
#define SX_ERR_TIMEOUT              10001   // Timeout
#define SX_ERR_NO_STATION           10002   // Cannot find the target unit
#define SX_ERR_FAIL_OPEN            10003   // Open failed
#define SX_ERR_NOT_OPEN             10004   // Not opened
#define SX_ERR_ALREADY_OPEN         10005   // Already opened
#define SX_ERR_NOT_CONTROL          10006   // Environment error
#define SX_ERR_ILLEGAL_PARAMETER    10007   // Invalid parameter
#define SX_ERR_SEND_ERROR           10008   // Send error
#define SX_ERR_RECV_ERROR           10009   // Receive error
#define SX_ERR_NOT_BLOCK            10010   // The received data is not in block
                                              format.
#define SX_ERR_SYSTEM_ERROR         10011   // System error
#define SX_ERR_ILLEGAL_ID           10012   // ID violation
#define SX_ERR_COMM_ERROR           10013   // Communication command error
#define SX_ERR_BUFFER_SHORT         10014   // Insufficient buffer
#define SX_ERR_NO_GROUP             10016   // Cannot find the target unit group
#define SX_ERR_ILLEGAL_GROUP        10017   // Invalid unit group
#define SX_ERR_HNDL_TYPE            10018   // Handle type violation
#define SX_ERR_ILLEGAL_HNDL         10019   // Handle error
#define SX_ERR_NO_HNDL              10020   // Cannot find handle
#define SX_ERR_ILLEAL_MESSAGE       10021   // Command string violation
#define SX_ERR_OUT_OF_RANG          10022   // Data outside of the range has been
                                              specified.
#define SX_ERR_NO_DATA              10023   // The specified data does not exist.
#define SX_ERR_CONFLICT             10024   // Conflict error
#define SX_ERR_INTERNAL_ERROR       10031   // Internal error
```

Unit Group Error Codes

```
#define SX_GRPERR_OK                0       // Group can be opened
#define SX_GRPERR_NOUNIT            1       // There are no units.
#define SX_GRPERR_MISSING           2       // The unit ID is missing. (The problem
                                              ID is b8 to b15).
#define SX_GRPERR_DUPLICATE         3       // There are duplicate IDs. (The
                                              problem ID is b8 to b15.)
```

Self Test Results

```
#define SX_SELFTEST_ACQMEM          0x0001  // Waveform memory error
#define SX_SELFTEST_SYSMEM          0x0002  // System memory error
#define SX_SELFTEST_BACKUPMEM       0x0004  // Backup memory error
#define SX_SELFTEST_HDD             0x0010  // Internal hard disk error
```

File Attribute

```
#define SX_ATTR_SUBDIR              0x0001  // Subdirectory
```

Trace Modes

```
#define SX_TRACE_FILE           0x00000001   // File output on
#define SX_TRACE_DEBUGOUT       0x00000002   // Debug output on (Only the
                                               debugging DLL is active.
                                               SX_TRACE_FILE has priority.)
#define SX_TRACE_SXAPI          0x00000010   // SxAPI.dll access log
#define SX_TRACE_SRQ            0x00000020   // SRQ processing log
#define SX_TRACE_TMCTL          0x00000040   // tmctl.dll access log
#define SX_TRACE_TIME           0x00000100   // Timestamp
#define SX_TRACE_ELASP          0x00000200   // Relative timestamp
#define SX_TRACE_PERFORM        0x00000400   // Performance measurement
#define SX_TRACE_ONLY_CALL      0x00001000   // Only trace function calls
#define SX_TRACE_ONLY_RETURN    0x00002000   // Only trace function returns
#define SX_TRACE_AUTOFILENAME   0x00010000   // Automatically insert the date,
                                               time, and a number after the
                                               file name (only valid when
                                               SX_TRACE_FILE is set to on)
```

4.1 Using Communication Commands

Functions for Sending and Receiving Communication Commands

Communication commands can be sent and received using the following SxAPI communication command control functions.

SxSetControl()	Sends a setup command
SxSetControlBinary()	Sends a setup command with the parameters in binary format
SxGetControl()	Sends and receives query commands
SxGetControlBinary()	Send and receives query commands with the parameters in binary format

For details about communication command control functions, see chapter 3.8.

Syntax Rules

- Parts of commands and parameters that are written in lowercase can be omitted.
  Example `:ALAR:ACK:EXEC`
- Units do not distinguish between uppercase and lowercase letters in commands and parameters.
  Example `alar:ack:exec`
- Parts of commands and parameters surrounded by square brackets ([ ]) can be omitted.
  Example `:TRIG:HYST MIDD`
- Parts of commands and parameters surrounded by curly brackets ({ })are variables. They cannot be omitted.
  Example `:CHAN<ch>:LSC:P1X 10`
- Vertical lines (|) indicate that a choice must be made from one of the items that they separate.
  Example `:ALAR:COMB OR`
- `<boolean>` indicates a boolean type. Boolean types are switched to true by 1 or on, and switched to false by 0 or off.
  Example `:GONO:ACT:BUZZ OFF`
- `<NRf>` indicates a numeric value.
  Example `:TRIG:LEV 2.5`
- `<Character string data>` indicates a string surrounded by double quotation marks.
  Example `:ALAR:ACT:MAIL:ADDR "yoko@yokogawa.jp.com"`
- `<Block data>` Indicates a binary format parameter.
- `<ch>` indicates a channel number. SxAPI will automatically produce the appropriate text, so you can just enter “`<ch>`” as-is.
  Example `:CHAN<ch>:LSC:P1X 10`
- `<mo>` indicates a module number. SxAPI will automatically produce the appropriate text, so you can just enter “mo” as-is.
- `<sg>` indicates a measuring group number. SxAPI will automatically produce the appropriate text, so you can just enter “sg” as-is.
- Other words surrounded by <> are indicative of the kinds of values that should be used in their place.

Note

Do not use commands that are not listed here (even if they are listed in IEEE 488.2-1987). Doing so may cause SxAPI to malfunction.

4.2 Commands

ALARm Group

Command	Description	Page
:ALARm:ACK:EXECute	Releases alarm ouput	4-6
:ALARm:ACOunt?	Queries the acquisition number, which is counted from the start of measurement, when an alarm occurs.	4-6
:ALARm:COMBination	Sets or queries the AND/OR state of the alarms of each channel.	4-6
:ALARm:CONDition?	Queries the alarm output terminal condition.	4-6
:ALARm:CHANnel<ch>:CONDition?	Queries the alarm condition of the specified channel.	4-6
:ALARm:CHANnel<ch>:HYSTeresis<X2>	Sets or queries the alarm hysteresis of a channel using three levels.	4-6
:ALARm:CHANnel<ch>:NHYSteresis<X2>	Sets or queries the alarm hysteresis of a channel using numeric values.	4-7
:ALARm:CHANnel<ch>:LEVel<X2>	Sets or queries the alarm level of a channel (when the input of the specified channel is not logic).	4-7
:ALARm:CHANnel<ch>:TYPE	Sets or queries the alarm type of a channel.	4-7
:ALARm:CHANnel<ch>:AVALue?	Queries the measured value at the alarm occurrence on the specified channels as an ASCII string.	4-7
:ALARm:CTIMe?	Queries the time of the most recent channel alarm condition change.	4-7
:ALARm:HOLD	Sets or queries alarm hold.	4-7
:ALARm:MODE	Sets or queries the alarm operation mode.	4-7
:ALARm:CMODe	Sets or queries the channel alarm operation mode.	4-7
:ALARm:SMODe	Sets or queries the system alarm operation mode.	4-7
:ALARm:OTERminal	Sets or queries the alarm output terminal on/off state.	4-7
:ALARm:SOURce	Sets or queries the alarm detection source.	4-7
:ALARm:STATus?	Queries the channel alarm status.	4-8
:ALARm:SSTATus?	Queries the system alarm status value.	4-8
:ALARm:SYSTem:SOURce:BOVerrun	Sets or queries the system alarm buffer overrun detection.	4-8
:ALARm:SYSTem:SOURce:FSTop	Sets or queries the system alarm fan stop detection.	4-8
:ALARm:SYSTem:SOURce:DFULl	Sets or queries the system alarm HDD full detection.	4-8
:ALARm:STIMe?	Queries the time of the most recent system alarm condition change.	4-8

CHANnel Group

Command	Description	Page
:CHANnel<ch>:ACCL:BIAS	Sets or queries the on/off state of the acceleration sensor's bias current (Acceleration/Voltage Module).	4-9
:CHANnel<ch>:ACCL:BWIDth	Sets or queries the filter (Acceleration/Voltage Module).	4-9
:CHANnel<ch>:ACCL:COUPling	Sets or queries input coupling (Acceleration/Voltage Module).	4-9
:CHANnel<ch>:ACCL:GAIN	Sets or queries the gain (Acceleration/Voltage Module).	4-9
:CHANnel<ch>:ACCL:SENSitivity	Sets or queries the sensitivity (Acceleration/Voltage Module).	4-9
:CHANnel<ch>:ACCL:UNIT	Sets or queries the unit of measurement of the upper and lower limit values (Acceleration/Voltage Module).	4-9
:CHANnel<ch>:FREQ:INPut:BWIDth	Sets or queries the bandwidth limit (Frequency Module).	4-9
:CHANnel<ch>:FREQ:INPut:CELimination	Sets or queries chattering elimination (Frequency Module).	4-9
:CHANnel<ch>:FREQ:INPut:COUPling	Sets or queries input coupling (Frequency Module).	4-10
:CHANnel<ch>:FREQ:INPut:HYSTeresis	Sets or queries hysteresis (Frequency Module).	4-10
:CHANnel<ch>:FREQ:INPut:PRESet	Sets or queries the preset (Frequency Module).	4-10
:CHANnel<ch>:FREQ:INPut:PROBe	Sets or queries the probe attenuation (Frequency Module).	4-10
:CHANnel<ch>:FREQ:INPut:PULLup	Sets or queries the pull-up on/off state (Frequency Module).	4-10
:CHANnel<ch>:FREQ:INPut:SLOPe	Sets or queries the slope (Frequency Module).	4-10
:CHANnel<ch>:FREQ:INPut:THReshold	Sets or queries the threshold level (Frequency Module).	4-10
:CHANnel<ch>:FREQ:INPut:VRANGe	Sets or queries the voltage range (Frequency Module).	4-10
:CHANnel<ch>:FREQ:LSCale:AVALue	Sets or queries linear scaling coefficient A (Frequency Module).	4-11
:CHANnel<ch>:FREQ:LSCale:BVALue	Sets or queries linear scaling coefficient B (Frequency Module).	4-11
:CHANnel<ch>:FREQ:LSCale:GETMeasure	Measures the X values of P1 and P2 for linear scaling (Frequency Module).	4-11
:CHANnel<ch>:FREQ:LSCale:MODE	Sets or queries linear scaling (Frequency Module).	4-11
:CHANnel<ch>:FREQ:LSCale:{P1X|P1Y|P2X|P2Y}	Sets or queries the X or Y value of P1 or P2 for linear scaling (Frequency Module).	4-11

Command	Description	Page
:CHANnel<ch>:FREQ:LSCale:UNIT	Sets or queries the unit of measurement to attach to the result of linear scaling (Frequency Module).	4-11
:CHANnel<ch>:FREQ:OFFSet	Sets or queries the offset value (Frequency Module).	4-11
:CHANnel<ch>:FREQ:SETup:CFRequency	Sets or queries the center frequency (Frequency Module).	4-12
:CHANnel<ch>:FREQ:SETup:DECeleration	Sets or queries the on/off state of decelerating prediction (Frequency Module).	4-12
:CHANnel<ch>:FREQ:SETup:DPULse	Sets or queries the distance per pulse (Frequency Module).	4-12
:CHANnel<ch>:FREQ:SETup:FILTer:SMOothing:MODE	Sets or queries the on/off state of smoothing (Frequency Module).	4-12
:CHANnel<ch>:FREQ:SETup:FILTer:SMOothing:VALue	Sets or queries the moving average order of smoothing (Frequency Module).	4-12
:CHANnel<ch>:FREQ:SETup:FILTer:PAVerage:MODE	Sets or queries the on/off state of pulse average (Frequency Module).	4-12
:CHANnel<ch>:FREQ:SETup:FILTer:PAVerage:VALue	Sets or queries the pulse average count (Frequency Module).	4-12
:CHANnel<ch>:FREQ:SETup:FUNCtion	Sets or queries the measuring mode (Frequency Module).	4-13
:CHANnel<ch>:FREQ:SETup:LRESet	Sets or queries the over limit reset (Frequency Module).	4-13
:CHANnel<ch>:FREQ:SETup:MPULse	Sets or queries whether the measurement pulse is positive or negative (Frequency Module).	4-13
:CHANnel<ch>:FREQ:SETup:PROTate	Sets or queries the number of pulses per rotation (Frequency Module).	4-13
:CHANnel<ch>:FREQ:SETup:RESet	Resets the pulse count (Frequency Module).	4-13
:CHANnel<ch>:FREQ:SETup:STOPpredict	Sets or queries the on/off state of stop prediction (Frequency Module).	4-13
:CHANnel<ch>:FREQ:SETup:TUNit	Sets or queries the time unit (Frequency Module).	4-13
:CHANnel<ch>:FREQ:SETup:UNIT	Sets or queries the pulse integration unit (Frequency Module).	4-13
:CHANnel<ch>:FREQ:SETup:UPULse	Sets or queries the unit/pulse (Frequency Module).	4-14
:CHANnel<ch>:FREQ:SETup:VUNit	Sets or queries the unit of velocity (Frequency Module).	4-14
:CHANnel<ch>:FREQ:VDIV	Sets or queries the Value/Div (Frequency Module).	4-14
:CHANnel<ch>:STRain:BALance:CHANnel<ch>	Sets or queries the channel on which balancing is to be executed (Strain Module).	4-14
:CHANnel<ch>:STRain:BALance:EXECute	Balances strain (Strain Module).	4-14
:CHANnel<ch>:STRain:BWIDth	Sets or queries the filter (Strain Module).	4-14
:CHANnel<ch>:STRain:EXCitation	Sets or queries the bridge voltage (Strain Module).	4-14
:CHANnel<ch>:STRain:GFACtor	Sets or queries the gauge factor (Strain Module).	4-14
:CHANnel<ch>:STRain:INVert	Sets or queries whether or not the display is inverted (Strain Module).	4-15
:CHANnel<ch>:STRain:LSCale:AVALue	Sets or queries linear scaling coefficient A (Strain Module).	4-15
:CHANnel<ch>:STRain:LSCale:BVALue	Sets or queries offset value B (Strain Module).	4-15
:CHANnel<ch>:STRain:LSCale:DISPlaytype:MODE	Sets or queries the display format for linear scaling (Strain Module).	4-15
:CHANnel<ch>:STRain:LSCale:DISPlaytype:DECimalnum	Sets or queries the decimal place when the display format for linear scaling is set to Floating (Strain Module).	4-15
:CHANnel<ch>:STRain:LSCale:GETMeasure	Measures the X values of P1 and P2 for linear scaling (Strain Module).	4-15
:CHANnel<ch>:STRain:LSCale:MODE	Sets or queries the linear scaling method (Strain Module).	4-15
:CHANnel<ch>:STRain:LSCale:{P1X|P1Y|P2X|P2Y}	Sets or queries the X or Y value of P1 or P2 for linear scaling (Strain Module).	4-16
:CHANnel<ch>:STRain:LSCale:SHUNt	Executes shunt calibration (Strain Module with DSUB, Shunt-Cal).	4-16
:CHANnel<ch>:STRain:LSCale:UNIT	Sets or queries the unit of measurement that is attached to the result of linear scaling (Strain Module).	4-16
:CHANnel<ch>:STRain:RANGe	Sets or queries the measuring range (Strain Module).	4-16
:CHANnel<ch>:STRain:UNIT	Sets or queries the unit of measurement (Strain Module).	4-16
:CHANnel<ch>:TEMPerature:BURNout	Sets or queries whether or not burnout is detected (Temperature, High Precision Voltage Isolation Module).	4-16
:CHANnel<ch>:TEMPerature:BWIDth	Sets or queries the bandwidth limit (Temperature, High Precision Voltage Isolation Module).	4-16
:CHANnel<ch>:TEMPerature:COUPling	Sets or queries input coupling (Temperature, High Precision Voltage Isolation Module).	4-16
:CHANnel<ch>:TEMPerature:RJC	Sets or queries the RJC (Temperature, High Precision Voltage Isolation Module).	4-17
:CHANnel<ch>:TEMPerature:TYPE	Sets or queries the thermocouple type (Temperature, High Precision Voltage Isolation Module).	4-17
:CHANnel<ch>:TEMPerature:UNIT	Sets or queries the unit of measurement values (Temperature, High Precision Voltage Isolation Module).	4-17

Command	Description	Page
:CHANnel<ch>[:VOLTage]:BWIDth	Sets or queries the bandwidth limit (voltage modules*).	4-17
:CHANnel<ch>[:VOLTage]:COUPling	Sets or queries input coupling (voltage modules*).	4-17
:CHANnel<ch>[:VOLTage]:INVert	Sets or queries the whether or not the display is inverted (voltage modules*).	4-17
:CHANnel<ch>[:VOLTage]:LSCale:BVALue	Sets or queries linear scaling offset value B (voltage modules*).	4-18
:CHANnel<ch>[:VOLTage]:LSCale:DISPlaytype:MODE	Sets or queries the display format for linear scaling.	4-18
:CHANnel<ch>[:VOLTage]:LSCale:DISPlaytype:DECimalnum	Sets or queries the decimal place when the display format for linear scaling is set to Floating.	4-18
:CHANnel<ch>[:VOLTage]:LSCale:DISPlaytype:SUBunit	Sets or queries the sub unit when the display format for linear scaling is set to Floating.	4-18
:CHANnel<ch>[:VOLTage]:LSCale:GETMeasure	Measures the X values of P1 and P2 for linear scaling (voltage modules*).	4-18
:CHANnel<ch>[:VOLTage]:LSCale:MODE	Sets or queries linear scaling (voltage modules*).	4-18
:CHANnel<ch>[:VOLTage]:LSCale:{P1X|P1Y|P2X|P2Y}	Sets or queries the X or Y value of P1 or P2 for linear scaling (voltage modules*).	4-19
:CHANnel<ch>[:VOLTage]:LSCale:UNIT	Sets or queries the unit of measurement to attach to the result of linear scaling (voltage modules*).	4-19
:CHANnel<ch>[:VOLTage]:PROBe	Sets or queries the probe type (voltage modules*).	4-19
:CHANnel<ch>[:VOLTage]:VDIV	Sets or queries the V/div value (voltage modules*).	4-19

* Voltage module refers to the High-Speed 100 MS/s, 12-Bit Isolation Module; the High-Speed 10 MS/s, 12-Bit Isolation Module; the High-Speed High-Resolution 1 MS/s, 16-Bit Isolation Module; the High-Speed 10 MS/s, 12-Bit Non-Isolation Module; and the High-Voltage 100 kS/s, 16-Bit Isolation Module (with RMS).

GONogo Group

Command	Description	Page
:GONogo:ACONdition	Sets or queries the GO/NO-GO judgment action condition.	4-20
:GONogo:ACTion:BUZZer	Sets or queries whether or not a beep is sounded when the condition is met.	4-20
:GONogo:ACTion:MAIL:ADDRess	Sets or queries the destination e-mail address for when the condition is met.	4-20
:GONogo:ACTion:MAIL:COUNt	Sets or queries the e-mail transmission limit for when the condition is met.	4-20
:GONogo:ACTion:MAIL:MODE	Sets or queries whether or not an e-mail is sent when the condition is met.	4-20
:GONogo:ACTion:SAVE[:MODE]	Sets or queries whether or not waveform data is saved to the storage media when the condition is met.	4-20
:GONogo:ACTion:SAVE:TYPE	Sets or queries the data type for saving waveform data to the storage media when the condition is met.	4-20
:GONogo:AREA	Sets or queries the waveform area that is judged.	4-20
:GONogo:COUNt?	Queries the number of performed GO/NO-GO judgments.	4-20
:GONogo:LOGic	Sets or queries the GO/NO-GO logical condition.	4-20
:GONogo:MODE	Sets or queries the GO/NO-GO judgment mode.	4-20
:GONogo:NGCount?	Queries the GO/NO-GO judgment NO-GO count.	4-21
:GONogo:PARameter:ITEM<x>:CAUSe?	Queries whether or not the specified waveform parameter is the cause of a NO-GO judgment.	4-21
:GONogo:PARameter:ITEM<x>:MODE	Sets or queries whether or not the specified waveform parameter is OFF, or what its judgment criterion is.	4-21
:GONogo:PARameter:ITEM<x>:PARam?	Queries the waveform parameter of the specified judgment condition.	4-21
:GONogo:PARameter:ITEM<x>:TRACe	Sets or queries the channel number of the specified judgment condition.	4-21
:GONogo:PARameter:ITEM<x>:TYPE:<Parameter>	Sets or queries the upper and lower limits of the judgment area for the specified judgment condition.	4-21
:GONogo:PARameter:ITEM<x>:VALue?	Queries the automated measurement value of the specified GO/NO-GO judgment parameter.	4-21
:GONogo:REMote	Sets or queries the GO/NO-GO judgment remote mode.	4-21
:GONogo:RSTatus?	Queries the most recent GO/NO-GO judgment.	4-21
:GONogo:SEQuence	Sets or queries the GO/NO-GO judgment action mode.	4-22
:GONogo:STOPcount	Sets or queries the GO/NO-GO judgment count.	4-22

MEASure Group

Command	Description	Page
:MEASure:AREA	Sets or queries the automatically measured waveform area for the waveform parameters.	4-23
:MEASure:CHANnel<ch>:DPRoximal:MODE	Sets or queries the distal, mesial, and proximal point mode setting.	4-24
:MEASure:CHANnel<ch>:DPRoximal:PERCent	Sets or queries the distal, mesial, and proximal points as percentages.	4-24
:MEASure:CHANnel<ch>:DPRoximal:UNIT	Sets or queries the distal, mesial, and proximal points.	4-24
:MEASure:CHANnel<ch>:METHod	Sets or queries the high/low point setting method.	4-24
:MEASure:CHANnel<ch>:<Parameter>:STATe	Sets or queries the on/off state of the measurement of the specified waveform parameter.	4-24
:MEASure:CHANnel<ch>:<Parameter>:VALue?	Queries the automated measurement value of the specified waveform parameter.	4-25
:MEASure:CRANge	Sets or queries the waveform parameter measurement range.	4-25
:MEASure:MODE	Sets or queries the waveform parameter automated measuring mode.	4-25
:MEASure:WAIT?	Waits for the execution of waveform parameter automated measurement with a set timeout.	4-25

TRIGger Group

Command	Description	Page
:TRIGger:COMBination:CHANnel<ch>:COMBination	Sets or queries the trigger AND/OR state of all of the bits in the specified channel for the combination trigger class.	4-26
:TRIGger:COMBination:CHANnel<ch>:HYSTeresis<x>	Sets or queries the trigger hysteresis of the specified channel in the combination trigger class.	4-26
:TRIGger:COMBination:CHANnel<ch>:LEVel<x>	Sets or queries the trigger level of the specified channel in the combination trigger class.	4-26
:TRIGger:COMBination:CHANnel<ch>:TYPE	Sets or queries the trigger type of the specified channel in the combination trigger class.	4-26
:TRIGger:COMBination:EXTernal:TYPE	Sets or queries the external trigger type of the specified channel in the combination trigger class.	4-26
:TRIGger:COMBination:MODE	Sets or queries the combination mode of the combination trigger class.	4-26
:TRIGger:DELay	Sets or queries the delay (the time between the trigger point and the trigger position)	4-27
:TRIGger:HOLDoff:TIME	Sets or queries the trigger hold off time.	4-27
:TRIGger[:SIMPle]:HYSTeresis	Sets or queries the hysteresis of the simple trigger.	4-27
:TRIGger[:SIMPle]:LEVel	Sets or queries the simple trigger of the channel specified by :TRIGger[:SIMPle]:SOURce.	4-27
:TRIGger[:SIMPle]:SLOPe	Sets or queries the simple trigger type of the channel specified by :TRIGger[:SIMPle]:SOURce.	4-28
:TRIGger:SIMPle:SOURce	Sets or queries the simple trigger source.	4-28
:TRIGger:TIMer:DATE	Sets or queries the date of the time trigger.	4-28
:TRIGger:TIMer:INTerval	Sets or queries the trigger interval of the time trigger.	4-28
:TRIGger:TIMer:TIME	Sets or queries the time of the time trigger.	4-28
:TRIGger:TYPE	Sets or queries the trigger type.	4-28

4.3 ALARm Group

ALArm group commands are only valid when the measuring mode is Free Run mode.

:ALARm:ACK:EXECute

Function Releases alarm output.
Syntax `:ALARm:ACK:EXECute`

:ALARm:ACOunt?

Function Queries the acquisition number, which is counted from the start of measurement, when an alarm occurs.
Syntax `:ALARm:ACOunt?`
Example `:ALARm:ACOunt? -> 65535`

:ALARm:COMBination

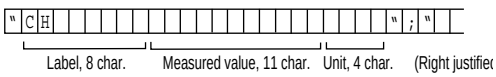
Function Sets or queries the AND/OR state of the alarms of each channel.
Syntax `:ALARm:COMBination {AND|OR}`
`:ALARm:COMBination?`
Example `:ALARm:COMBination OR`
`:ALARm:COMBination? -> OR`

:ALARm:CONDition?

Function Queries the alarm output terminal condition.
Syntax `:ALARm:CONDition?`
Example `:ALARm:CONDition? -> 1`
Description If output is on, the command returns 1. If output is off, the command returns 0.

:ALARm:CHANnel<ch>:AVALue?

Function Queries the measured value at the alarm occurrence on the specified channels as an ACII string.
Syntax `:ALARm:CHANnel<ch>:AVALue?`
Description The measured values of each channel are divided by semicolons (0x3b). The values for all active channels are output.
You can use the VERBose setting to decide whether or not returned values have labels and units.
If a channels display is set to Off, its measured value will be listed as “Off.”
Format:



:ALARm:CHANnel<ch>:CONDition?

Function Queries the alarm condition of the specified channel.
Syntax `:ALARm:CHANnel<ch>:CONDition?`
Example `:ALARm:CHANnel<ch>:CONDition? -> 1`
Description If the alarm is on, the command returns 1. If the alarm is off, the command returns 0.

:ALARm:CHANnel<ch>:HYSTeresis<X2>

Function Sets or queries the alarm hysteresis of a channel using three levels.
Syntax `:ALARm:CHANnel<ch>:HYSTeresis<X2>`
`{HIGH|LOW|MIDDle}`
`:ALARm:CHANnel<ch>:HYSTeresis<X2>?`
<X2> = 1, 2
If TYPE is HIGH, LOW, only level 1 is used.
If TYPE is WLIn, WLOut, both level 1 and 2 are used.
Level 1 is the upper limit. Level 2 is the lower limit.
HIGH: ±10% (of 10 times V/Div)
MIDDle: ±5% (of 10 times V/Div)
LOW: ±1% (of 10 times V/Div)
Example `:ALARm:CHANnel<ch>:HYSTeresis1 LOW`
`:ALARm:CHANnel<ch>:HYSTeresis1? -> LOW`

:ALARm:CHANnel<ch>:NHYSteresis<X2>

Function Sets or queries the alarm hysteresis of a channel using numeric values.

Syntax `:ALARm:CHANnel<ch>:NHYSteresis<X2> {<Voltage>|<NRf>|<Current>}`
`:ALARm:CHANnel<ch>:NHYSteresis<X2>?`
<X2> = 1, 2
If TYPE is HIGH, LOW, only level 1 is used.
If TYPE is WLIn, WLOut, both level 1 and 2 are used.
Level 1 is the upper limit. Level 2 is the lower limit.

Example `:ALARm:CHANnel<ch>:NHYSteresis2 2.0`
`:ALARm:CHANnel<ch>:NHYSteresis2? -> 2.000E+00`

:ALARm:CHANnel<ch>:LEVel<X2>

Function Sets or queries the alarm level of a channel (when the input of the specified channel is not logic).

Syntax `:ALARm:CHANnel<ch>:LEVel<X2> {<Voltage>|<NRf>|<Current>}`
`:ALARm:CHANnel<ch>:LEVel<X2>?`
<X2> = 1, 2
If TYPE is HIGH, LOW, only level 1 is used.
If TYPE is WLIn, WLOut, both level 1 and 2 are used.
Level 1 is the upper limit. Level 2 is the lower limit.

Description The Au7Fe temperature measuring range is 0 to 280 K (-273 to 7°C)

:ALARm:CHANnel<ch>:TYPE

Function Sets or queries the alarm type of a channel.

Syntax `:ALARm:CHANnel<ch>:TYPE {HIGH|LOW|OFF|WLIn|WLOut}`
`:ALARm:CHANnel<ch>:TYPE?`

:ALARm:CTIMe?

Function Queries the time of the most recent channel alarm condition change.

Syntax `:ALARm:CTIMe? -> {<NRf>,<NRf>,<NRf>,<NRf>,<NRf>,<NRf>,<NRf>}`

<NRf>	Year	(from 2007)
<NRf>	Month	(1 to 12)
<NRf>	Day	(1 to 31)
<NRf>	Hour	(0 to 23)
<NRf>	Minute	(0 to 59)
<NRf>	Second	(0 to 59)
<NRf>	Millisecond	(0 to 900)

Description If no status changes occur after measurement starts, the function will return the time when measurement started. If measurement has not started, the function will return an undefined value.

:ALARm:HOLD

Function Sets or queries alarm hold.

Syntax `:ALARm:HOLD {<boolean>}`
`:ALARm:HOLD?`

Description If the alarm hold is on, alarm output will continue until ALAR:ACK:EXEC is received even if the condition that caused the alarm is cleared.

:ALARm:MODE

Function Sets or queries the alarm operation mode.

Syntax `:ALARm:MODE {OFF|ON}`
`:ALARm:MODE?`
Off: The SL1000 does not output alarms.
On: The SL1000 detects alarms during measurement (if alarm hold is on, the alarm will continue to be output even after measurement stops).

:ALARm:CMODe

Function Sets or queries the channel alarm operation mode.

Syntax `:ALARm:CMODe {<boolean>}`
`:ALARm:CMOD?`

Description Sets whether to detect or not detect the channel alarm.

:ALARm:SMODe

Function Sets or queries the system alarm operation mode.

Syntax `:ALARm:SMODe {<boolean>}`
`:ALARm:SMODe?`

Description Sets whether to detect or not detect the system alarm.

:ALARm:OTERminal

Function Sets or queries the alarm output terminal on/off state.

Syntax `:ALARm:OTERminal {<boolean>}`
`:ALARm:OTERminal?`

Description If the alarm output terminal is switched off, it will remain off regardless of the alarm settings or conditions.

:ALARm:SOURce

Function Sets or queries the alarm detection source.

Syntax `:ALARm:SOURce {CHANnel|SYSTem}`
`:ALARm:SOURce?`

Description The initial value is CHANnel.

:ALARm:STATus?

Function Queries the channel alarm status.

Syntax :ALARm:STATus?

Example :ALARm:STATus? -> 65535

Description The :ALARm:CHANnel<ch>:CONDition? command must be used repeatedly to acquire the alarm statuses of all channels, so this command returns a bit pattern.
The MSB of the returned value (bit 15) represents channel 1. The LSB (bit 0) represents channel 16.
A bit value of 1 represents an active alarm status.
The channel numbers referred to here are physical channel numbers, which are different from the actual channel numbers.

:ALARm:SSTATus?

Function Queries the system alarm status value.

Syntax :ALARm:SSTATus?

Example :ALARm:SSTATus? -> 2

Description The system status bit assignments are shown in the table below.

Bit	Name	Description
0	Reserved	
1	HDD_FULL	The disk is full.
2	FAN_STOP	The fan has stopped.
3	Reserved	
4	BUF_OVERRUN_HOST	Host (PC) buffer has overrun.
5	BUF_OVERRUN_UNIT	The SL1000 buffer has overrun.
6 to 15	Reserved	

:ALARm:SYSTem:SOURce:BOVerrun

Function Sets or queries the system alarm buffer overrun detection.

Syntax :ALARm:SYSTem:SOURce:
BOVerrun {<boolean>}
:ALARm:SYSTem:SOURce:BOVerrun?

:ALARm:SYSTem:SOURce:FSTop

Function Sets or queries the system alarm fan stop detection.

Syntax :ALARm:SYSTem:SOURce:FSTop {<boolean>}
:ALARm:SYSTem:SOURce:FSTop?

:ALARm:SYSTem:SOURce:DFULl

Function Sets or queries the system alarm HDD full detection.

Syntax :ALARm:SYSTem:SOURce:DFULl {<boolean>}
:ALARm:SYSTem:SOURce:DFULl?

:ALARm:STIMe?

Function Queries the time of the most recent system alarm condition change.

Syntax :ALARm:STIMe? -> {<NRf>,<NRf>,<NRf>,
<NRf>,<NRf>,<NRf>,<NRf>}

<NRf>	Year	(from 2007)
<NRf>	Month	(1 to 12)
<NRf>	Day	(1 to 31)
<NRf>	Hour	(0 to 23)
<NRf>	Minute	(0 to 59)
<NRf>	Second	(0 to 59)
<NRf>	Millisecond	(0 to 900)

Description If no status changes occur after measurement starts, the function will return the time when measurement started.

4.4 CHANnel Group

The CHANnel group functions deal with the vertical axes of the SL1000 channels.

:CHANnel<ch>:ACCL:BIAS

Function Sets or queries the on/off status of the acceleration sensor's bias current when an Acceleration/Voltage Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:ACCL:BIAS {<boolean>}`
`:CHANnel<ch>:ACCL:BIAS?`

Example `:CHANNEL<ch>:ACCL:BIAS ON`
`:CHANNEL<ch>:ACCL:BIAS? -> 1`

Description An error occurs if an Acceleration/Voltage Module is not installed.

:CHANnel<ch>:ACCL:BWIDth

Function Sets or queries the filter when an Acceleration/Voltage Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:ACCL:BWIDth {FULL|AUTO|<Frequency>}`
`:CHANnel<ch>:ACCL:BWIDth?`
<Frequency> = 4 kHz, 400 Hz, 40 Hz

Example `:CHANNEL<ch>:ACCL:BWIDTH AUTO`
`:CHANNEL<ch>:ACCL:BWIDTH? -> AUTO`

Description An error occurs if an Acceleration/Voltage Module is not installed.

:CHANnel<ch>:ACCL:COUPling

Function Sets or queries input coupling when an Acceleration/Voltage Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:ACCL:COUPling {AC|DC|ACCL|GND}`
`:CHANnel<ch>:ACCL:COUPling?`

Example `:CHANNEL<ch>:ACCL:COUPLING GND`
`:CHANNEL<ch>:ACCL:COUPLING? -> GND`

Description An error occurs if an Acceleration/Voltage Module is not installed.

:CHANnel<ch>:ACCL:GAIN

Function Sets or queries the gain when an Acceleration/Voltage Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:ACCL:GAIN {<NRf>}`
`:CHANnel<ch>:ACCL:GAIN?`
<NRf> = 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100

Example `:CHANNEL<ch>:ACCL:GAIN 100`
`:CHANNEL<ch>:ACCL:GAIN? -> 100.0`

Description An error occurs if an Acceleration/Voltage Module is not installed.

:CHANnel<ch>:ACCL:SENSitivity

Function Sets or queries the sensitivity when an Acceleration/Voltage Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:ACCL:SENSitivity {<NRf>}`
`:CHANnel<ch>:ACCL:SENSitivity?`
<NRf> = 0.1 to 2000

Example `:CHANNEL<ch>:ACCL:SENSITIVITY 10`
`:CHANNEL<ch>:ACCL:SENSITIVITY? -> 10.00`

Description An error occurs if an Acceleration/Voltage Module is not installed.

:CHANnel<ch>:ACCL:UNIT

Function Sets or queries the unit of measurement of the upper and lower limit values when an Acceleration/Voltage Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:ACCL:UNIT {<String>}`
`:CHANnel<ch>:ACCL:UNIT?`

Example `:CHANNEL<ch>:ACCL:UNIT "ACCL"`
`:CHANNEL<ch>:ACCL:UNIT? -> "ACCL"`

Description An error occurs if an Acceleration/Voltage Module is not installed.

:CHANnel<ch>:FREQ:INPut:BWIDth

Function Sets or queries the bandwidth limit when a Frequency Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:FREQ:INPut:BWIDth {FULL|<Frequency>}`
`:CHANnel<ch>:FREQ:INPut:BWIDth?`
<Frequency> = 100 Hz, 1 kHz, 10 kHz, 100 kHz

Example `:CHANNEL<ch>:FREQ:INPUT:BWIDTH FULL`
`:CHANNEL<ch>:FREQ:INPUT:BWIDTH? -> FULL`

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPut:CELimination

Function Sets or queries chattering elimination when a Frequency Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:FREQ:INPut:CELimination{<Time>}`
`:CHANnel<ch>:FREQ:INPut:CELimination?`
<Time> = 0 to 1000 ms

Example `:CHANNEL<ch>:FREQ:INPUT:CELIMINATION 100ms`
`:CHANNEL<ch>:FREQ:INPUT:CELIMINATION? -> 0.100`

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPut:COUPling

Function Sets or queries input coupling when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:INPut:
COUPling {AC|DC}
:CHANnel<ch>:FREQ:INPut:COUPling?
```

Example
```
:CHANNEL<ch>:FREQ:INPUT:COUPLING DC
:CHANNEL<ch>:FREQ:INPUT:COUPLING? -> DC
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPut:HYSTeresis

Function Sets or queries hysteresis when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:INPut:
HYSTeresis{HIGH|LOW|MIDDle}
:CHANnel<ch>:FREQ:INPut:HYSTeresis?
```

Example
```
:CHANNEL<ch>:FREQ:INPUT:HYSTERESIS LOW
:CHANNEL<ch>:FREQ:INPUT:HYSTERESIS?
-> LOW
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPut:PRESet

Function Sets or queries the preset when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:INPut:PRESet {AC100v|
AC200v|EMPichup|LOG12v|LOG24v|LOG3v|
LOG5v|PULLup|USER|ZERO}
:CHANnel<ch>:FREQ:INPut:PRESet?
```

Example
```
:CHANNEL<ch>:FREQ:INPUT:PRESET USER
:CHANNEL<ch>:FREQ:INPUT:PRESET? -> USER
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPut:PROBe

Function Sets or queries the probe attenuation when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:INPut:PROBe {<NRf>}
:CHANnel<ch>:FREQ:INPut:PROBe?
```
<NRf> = 1, 10

Example
```
:CHANNEL<ch>:FREQ:INPUT:PROBE 10
:CHANNEL<ch>:FREQ:INPUT:PROBE? -> 10
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPut:PULLup

Function Sets or queries the pull-up on/off state when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:INPut:
PULLup {<boolean>}
:CHANnel<ch>:FREQ:INPut:PULLup?
```

Example
```
:CHANNEL<ch>:FREQ:INPUT:PULLUP ON
:CHANNEL<ch>:FREQ:INPUT:PULLUP? -> 1
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPut:SLOPe

Function Sets or queries the slope when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:INPut:
SLOPe {FALL|RISE}
:CHANnel<ch>:FREQ:INPut:SLOPe?
```

Example
```
:CHANNEL<ch>:FREQ:INPUT:SLOPE FALL
:CHANNEL<ch>:FREQ:INPUT:SLOPE? -> FALL
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPut:THReshold

Function Sets or queries the threshold level when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:INPut:
THReshold {<Voltage>}
:CHANnel<ch>:FREQ:INPut:THReshold?
```
<Voltage> = 100 to –100 V

Example
```
:CHANNEL<ch>:FREQ:INPUT:THRESHOLD 10
:CHANNEL<ch>:FREQ:INPUT:THRESHOLD?
-> 10.000E+00
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPut:VRANGe

Function Sets or queries the voltage range when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:INPut:
VRANGe {<Voltage>}
:CHANnel<ch>:FREQ:INPut:VRANGe?
```
<Voltage> = 1 to 500 V

Example
```
:CHANNEL<ch>:FREQ:INPUT:VRANGE 10
:CHANNEL<ch>:FREQ:INPUT:VRANGE? -> 10
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:LSCale:AVALue

Function Sets or queries linear scaling coefficient A when a Frequency Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:FREQ:LSCale:AVALue {<NRf>}`
`:CHANnel<ch>:FREQ:LSCale:AVALue?`
<NRf> = –9.9999E + 30 to 9.9999E+30

Example `:CHANNEL<ch>:FREQ:LSCALE:AVALUE 10`
`:CHANNEL<ch>:FREQ:LSCALE:AVALUE?`
`-> 10.0000E+00`

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:LSCale:BVALue

Function Sets or queries linear scaling coefficient B when a Frequency Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:FREQ:LSCale:BVALue {<NRf>}`
`:CHANnel<ch>:FREQ:LSCale:BVALue?`
<NRf> = –9.9999E+30 to 9.9999E+30

Example `:CHANNEL<ch>:FREQ:LSCALE:BVALUE 10`
`:CHANNEL<ch>:FREQ:LSCALE:BVALUE?`
`-> 10.0000E+00`

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:LSCale:GETMeasure

Function Measures the X values of P1 and P2 for linear scaling when a Frequency Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:FREQ:LSCale:`
`GETMeasure {P1X|P2X}`

Example `:CHANNEL<ch>:FREQ:LSCALE:GETMEASURE P1X`

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:LSCale:MODE

Function Sets or queries linear scaling when a Frequency Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:FREQ:LSCale:`
`MODE {AXB|OFF|P12}`
`:CHANnel<ch>:FREQ:LSCale:MODE?`

Example `:CHANNEL<ch>:FREQ:LSCALE:MODE OFF`
`:CHANNEL<ch>:FREQ:LSCALE:MODE? -> OFF`

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:LSCale:{P1X|P1Y|P2X|P2Y}

Function Sets or queries the X or Y value of P1 or P2 for linear scaling when a Frequency Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:FREQLSCale:{P1X|P1Y|P2X|`
`P2Y} {<NRf>}`
`:CHANnel<ch>:FREQ:LSCale:{P1X|P1Y|P2X|`
`P2Y}?`
For P1X and P2X, **<NRf>** = –9.9999E+30 to 9.9999E+30
For P1Y and P2Y, **<NRf>** = –9.9999E+25 to 9.9999E+25

Example `:CHANNEL<ch>:FREQ:LSCALE:P1X 10`
`:CHANNEL<ch>:FREQ:LSCALE:P1X?`
`-> 10.0000E+00`

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:LSCale:UNIT

Function Sets or queries the unit of measurement to attach to the result of linear scaling when a Frequency Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:FREQ:LSCale:`
`UNIT {<String>}`
`:CHANnel<ch>:FREQ:LSCale:UNIT?`
<String> = Up to 4 characters

Example `:CHANNEL<ch>:FREQ:LSCALE:UNIT "AAA"`
`:CHANNEL<ch>:FREQ:LSCALE:UNIT? -> "AAA"`

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:OFFSet

Function Sets or queries the offset value when a Frequency Module is installed in the specified channel (slot).

Syntax `:CHANnel<ch>:FREQ:OFFSet {<NRf>|`
`<Frequency>|<Time>}`
`:CHANnel<ch>:FREQ:OFFSet?`
{<NRf>|<Frequency>|<Time>} = The settable range varies depending on the range setting. For details, see the *SL1000 High Speed Data Acquisition Unit User's Manual*.

Example `:CHANNEL<ch>:FREQ:OFFSET 1`
`:CHANNEL<ch>:FREQ:OFFSET?`
`-> 1.000000E+00`

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:CFRequency

Function Sets or queries the center frequency when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:
CFRequency {<Frequency>}
:CHANnel<ch>:FREQ:SETup:CFRequency?
```
<NRf> = 50 Hz, 60 Hz, 400 Hz

Example
```
:CHANNEL<ch>:FREQ:SETUP:CFREQUENCY 50
:CHANNEL<ch>:FREQ:SETUP:CFREQUENCY?
-> 50
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:DECeleration

Function Sets or queries the on/off state of decelerating prediction when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:
DECeleration {<boolean>}
:CHANnel<ch>:FREQ:SETup:DECeleration?
```

Example
```
:CHANNEL<ch>:FREQ:SETUP:DECELERATION ON
:CHANNEL<ch>:FREQ:SETUP:DECELERATION?
-> 1
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:DPULse

Function Sets or queries the distance per pulse when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:DPULse {<NRf>}
:CHANnel<ch>:FREQ:SETup:DPULse?
```
<NRf> = 9.9999E+30 to –9.9999E+30

Example
```
:CHANNEL<ch>:FREQ:SETUP:DPULSE 1e15
:CHANNEL<ch>:FREQ:SETUP:DPULSE?
-> 1.00000E+15
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:FILTer:SMOothing:MODE

Function Sets or queries the on/off state of smoothing when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:FILTer:
SMOothing:MODE {<boolean>}
:CHANnel<ch>:FREQ:SETup:FILTer:
SMOothing:MODE?
```

Example
```
:CHANNEL<ch>:FREQ:SETUP:FILTER:
SMOOTHING:MODE ON
:CHANNEL<ch>:FREQ:SETUP:FILTER:
SMOOTHING:MODE?  -> 1
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:FILTer:SMOothing:VALue

Function Sets or queries the moving average order of smoothing when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:FILTer:
SMOothing:VALue {<Time>}
:CHANnel<ch>:FREQ:SETup:FILTer:
SMOothing:VALue?
```
{**<Time>**} = 0 to 1000

Example
```
:CHANNEL<ch>:FREQ:SETUP:FILTER:
SMOOTHING:VALUE 10ms
:CHANNEL<ch>:FREQ:SETUP:FILTER:
SMOOTHING:VALUE? -> 0.0100
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:FILTer:PAVerage:MODE

Function Sets or queries the on/off state of pulse average when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:FILTer:
PAVerage:MODE {<boolean>}
:CHANnel<ch>:FREQ:SETup:FILTer:
PAVerage:MODE?
```

Example
```
:CHANNEL<ch>:FREQ:SETUP:FILTER:
PAVERAGE:MODE ON
:CHANNEL<ch>:FREQ:SETUP:FILTER:
PAVERAGE:MODE? -> 1
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:FILTer:PAVerage:VALue

Function Sets or queries the pulse average count when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:FILTer:
PAVerage:VALue {<NRf>}
:CHANnel<ch>:FREQ:SETup:FILTer:
PAVerage:VALue?
```
<NRf> = 1 to 4096

Example
```
:CHANNEL<ch>:FREQ:SETUP:FILTER:
PAVERAGE:VALUE 10
:CHANNEL<ch>:FREQ:SETUP:FILTER:
PAVERAGE:VALUE? -> 10
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:FUNCtion

Function Sets or queries the measuring mode when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:
FUNCtion{FREQuency|RPM|RPS|PERiod|
DUTY|PWIDth|PINTeg|VELocity}
:CHANnel<ch>:FREQ:SETup:FUNCtion?
```

Example
```
:CHANNEL<ch>:FREQ:SETUP:
FUNCTION FREQUENCY
:CHANNEL<ch>:FREQ:SETUP:FUNCTION?
-> FREQUENCY
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:LRESet

Function Sets or queries the over limit reset when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:
LRESet{<boolean>}
:CHANnel<ch>:FREQ:SETup:LRESet?
```

Example
```
:CHANNEL<ch>:FREQ:SETUP:LRESET ON
:CHANNEL<ch>:FREQ:SETUP:LRESET? -> 1
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:MPULse

Function Sets or queries whether the measurement pulse is positive or negative when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:
MPULse{POSitive|NEGative}
:CHANnel<ch>:FREQ:SETup:MPULse?
```

Example
```
:CHANNEL<ch>:FREQ:SETUP:MPULSE POSITIVE
:CHANNEL<ch>:FREQ:SETUP:MPULSE?
-> POSITIVE
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:PROTate

Function Sets or queries the number of pulses per rotation when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:PROTate {<NRf>}
:CHANnel<ch>:FREQ:SETup:PROTate?
```
<NRf> = 1 to 99999

Example
```
:CHANNEL<ch>:FREQ:SETUP:PROTATE 10
:CHANNEL<ch>:FREQ:SETUP:PROTATE? -> 10
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:RESet

Function Resets the pulse count when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:RESet
```

Example
```
:CHANNEL<ch>:FREQ:SETUP:RESET
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:STOPpredict

Function Sets or queries the on/off state of stop prediction when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:
STOPpredict{<NRf>|OFF}
:CHANnel<ch>:FREQ:SETup:STOPpredict?
```
<NRf> = 1.5, 2, 3, 4, 5, 6, 7, 8, 9, 10

Example
```
:CHANNEL<ch>:FREQ:SETUP:STOPPREDICT OFF
:CHANNEL<ch>:FREQ:SETUP:STOPPREDICT?
-> OFF
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:TUNit

Function Sets or queries the time unit when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:
TUNit {HOUR|MIN|SEC}
:CHANnel<ch>:FREQ:SETup:TUNit?
```

Example
```
:CHANNEL<ch>:FREQ:SETUP:TUNIT SEC
:CHANNEL<ch>:FREQ:SETUP:TUNIT? -> SEC
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:UNIT

Function Sets or queries the pulse integration unit when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:UNIT {<String>}
:CHANnel<ch>:FREQ:SETup:UNIT?
```
<String> = Up to 4 characters

Example
```
:CHANNEL<ch>:FREQ:SETUP:UNIT "AAA"
:CHANNEL<ch>:FREQ:SETUP:UNIT? -> "AAA"
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:UPULse

Function Sets or queries the unit/pulse when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:UPULse {<NRf>}
:CHANnel<ch>:FREQ:SETup:UPULse?
```
<NRf> = 9.9999E+30 to -9.9999E+30

Example
```
:CHANNEL<ch>:FREQ:SETUP:UPULSE 1e15
:CHANNEL<ch>:FREQ:SETUP:UPULSE?
-> 1.00000E+15
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:VUNit

Function Sets or queries the unit of velocity when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:SETup:VUNit {<String>}
:CHANnel<ch>:FREQ:SETup:VUNit?
```
<String> = Up to 4 characters

Example
```
:CHANNEL<ch>:FREQ:SETUP:VUNIT "BBB"
:CHANNEL<ch>:FREQ:SETUP:VUNIT? -> "BBB"
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:VDIV

Function Sets or queries the Value/Div when a Frequency Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:FREQ:
VDIV {<NRf>|<Frequency>|<Time>}
:CHANnel<ch>:FREQ:VDIV?
```
{**<NRf>**|**<Frequency>**|**<Time>**} = See the *SL1000 High Speed Data Acquisition Unit User's Manual* for details.

Example
```
:CHANNEL<ch>:FREQ:VDIV 10
:CHANNEL<ch>:FREQ:VDIV? -> 10.0E+00
```

Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:STRain:BALance:CHANnel<ch>

Function Sets or queries the channel on which balancing is to be executed when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:BALance:
CHANnel<ch>{<boolean>}
:CHANnel<ch>:STRain:BALance:CHANnel?
```

Example
```
:CHANNEL<ch>:STRAIN:BALANCE:
CHANNEL<ch> ON
:CHANNEL<ch>:STRAIN:BALANCE:
CHANNEL<ch>? -> 1
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:BALance:EXECute

Function Balances strain when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:BALance:EXECute
```

Example
```
:CHANNEL<ch>:STRAIN:BALANCE:EXECUTE
```

Description
- Balances channels that are switched on with the: CHANnel<ch>:STRain:BALance:CHANnel<ch> command.
- An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:BWIDth

Function Sets or queries the filter when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:BWIDth {FULL|
<Frequency>}
:CHANnel<ch>:STRain:BWIDth?
```
<Frequency> = 10 Hz, 100 Hz, 1 kHz

Example
```
:CHANNEL<ch>:STRAIN:BWIDTH FULL
:CHANNEL<ch>:STRAIN:BWIDTH? -> FULL
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:EXCitation

Function Sets or queries the bridge voltage when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:
EXCitation {<Voltage>}
:CHANnel<ch>:STRain:EXCitation?
```
<Voltage> = 2 V, 5 V, 10 V

Example
```
:CHANNEL<ch>:STRAIN:EXCITATION 2V
:CHANNEL<ch>:STRAIN:EXCITATION?
-> 2.000000E+00
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:GFACtor

Function Sets or queries the gauge factor when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:GFACtor {<NRf>}
:CHANnel<ch>:STRain:GFACtor?
```
<NRf> = 1.90 to 2.20

Example
```
:CHANNEL<ch>:STRAIN:GFACTOR 2.00
:CHANNEL<ch>:STRAIN:GFACTOR? -> 2.00
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:INVert

Function Sets or queries whether or not the display is inverted when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:INVert {<boolean>}
:CHANnel<ch>:STRain:INVert?
```

Example
```
:CHANNEL<ch>:STRAIN:INVERT ON
:CHANNEL<ch>:STRAIN:INVERT? -> 1
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:AVALue

Function Sets or queries linear scaling coefficient A when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:LSCale:
AVALue {<NRf>}
:CHANnel<ch>:STRain:LSCale:AVALue?
```
<NRf> = –9.9999E+30 to 9.9999E+30

Example
```
:CHANNEL<ch>:STRAIN:LSCALE:AVALUE 10
:CHANNEL<ch>:STRAIN:LSCALE:AVALUE?
-> 10.0000E+00
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:BVALue

Function Sets or queries offset value B when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:LSCale:
BVALue {<NRf>}
:CHANnel<ch>:STRain:LSCale:BVALue?
```
<NRf> = –9.9999E+30 to 9.9999E+30

Example
```
:CHANNEL<ch>:STRAIN:LSCALE:BVALUE 5
:CHANNEL<ch>:STRAIN:LSCALE:BVALUE?
-> 5.00000E+00
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:DISPlaytype:MODE

Function Sets or queries the display format for linear scaling.

Syntax
```
:CHANnel<ch>:STRain:LSCale:DISPlaytype:
MODE {EXPonent|FLOating}
:CHANnel<ch>:STRain:LSCale:DISPlaytype:
MODE?
```

Example
```
:CHANNEL<ch>:STRAIN:LSCALE:DISPLAYTYPE:
MODE EXPONENT
:CHANNEL<ch>:STRAIN:LSCALE:DISPLAYTYPE:
MODE? -> EXPONENT
```

:CHANnel<ch>:STRain:LSCale:DISPlaytype:DECimalnum

Function Sets or queries the decimal place when the display format for linear scaling is set to Floating.

Syntax
```
:CHANnel<ch>:STRain:LSCale:
DISPlaytype:DECimalnum {<NRf>|AUTO}
:CHANnel<ch>:STRain:LSCale:
DISPlaytype:DECimalnum?
```
<NRf> = 0 to 3

Example
```
:CHANNEL<ch>:STRAIN:LSCALE:
DISPLAYTYPE:DECIMALNUM AUTO
:CHANNEL<ch>:STRAIN:LSCALE:
DISPLAYTYPE:DECIMALNUM? -> AUTO
```

:CHANnel<ch>:STRain:LSCale:GETMeasure

Function Measures the X values of P1 and P2 for linear scaling when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:LSCale:
GETMeasure {P1X|P2X}
```

Example
```
:CHANNEL<ch>:STRAIN:LSCALE:
GETMeasure P1X
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:MODE

Function Sets or queries the linear scaling method when a Strain Module is installed in the specified channel (slot). (The method can only be set to SHUNt with a Strain Module with DSUB, Shunt-Cal.)

Syntax
```
:CHANnel<ch>:STRain:LSCale:MODE {AXB|
OFF|P12|SHUNt}
:CHANnel<ch>:STRain:LSCale:MODE?
```

Example
```
:CHANNEL<ch>:STRAIN:LSCALE:MODE AXB
:CHANNEL<ch>:STRAIN:LSCALE:MODE?
-> AXB
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:{P1X|P1Y|P2X|P2Y}

Function Sets or queries the X or Y value of P1 or P2 for linear scaling when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:LSCale:
{P1X|P1Y|P2X|P2Y} {<NRf>}
:CHANnel<ch>:STRain:LSCale:
{P1X|P1Y|P2X|P2Y}?
```
For P1X and P2X, **<NRf>** = –9.9999E+30 to 9.9999E+30
For P1Y and P2Y, **<NRf>** = –9.9999E+25 to 9.9999E+25

Example
```
:CHANNEL<ch>:STRAIN:LSCALE:P1X 10
:CHANNEL<ch>:STRAIN:LSCALE:P1X?
-> 10.0000E+00
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:SHUNt

Function Executes shunt calibration when a Strain Module is installed in the specified channel (slot). (This command only works with a Strain Module with DSUB, Shunt-Cal.)

Syntax
```
:CHANnel<ch>:STRain:LSCale:SHUNt
```

Example
```
:CHANNEL<ch>:STRAIN:LSCALE:SHUNT
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:UNIT

Function Sets or queries the unit of measurement to attach to the result of linear scaling when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:LSCale:
UNIT {<String>}
:CHANnel<ch>:STRain:LSCale:UNIT?
```
<String> = Up to 4 characters

Example
```
:CHANNEL<ch>:STRAIN:LSCALE:UNIT "X"
:CHANNEL<ch>:STRAIN:LSCALE:UNIT? -> "X"
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:RANGe

Function Sets or queries the measuring range when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:RANGe {<NRf>}
:CHANnel<ch>:STRain:RANGe?
```
<NRf> = 0.25, 0.5, 1, 2.5, 5, 10 (mV/V)
500, 1000, 2000, 5000, 10000, 20000 (μSTR)

Example
```
:CHANNEL<ch>:STRAIN:RANGE 5000
:CHANNEL<ch>:STRAIN:RANGE? -> 5000
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:UNIT

Function Sets or queries the unit of measurement when a Strain Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:STRain:UNIT {MV|USTR}
:CHANnel<ch>:STRain:UNIT?
```

Example
```
:CHANNEL<ch>:STRAIN:UNIT USTR
:CHANNEL<ch>:STRAIN:UNIT? -> USTR
```

Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>:TEMPerature:BURNout

Function Sets or queries whether or not burnout is detected when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:TEMPerature:
BURNout {<boolean>}
```

Example
```
:CHANNEL<ch>:TEMPERATURE:BURNOUT ON
:CHANNEL<ch>:TEMPERATURE:BURNOUT? -> 1
```

Description An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

:CHANnel<ch>:TEMPerature:BWIDth

Function Sets or queries the bandwidth limit when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:TEMPerature:BWIDth {FULL|
<Frequency>}
:CHANnel<ch>:TEMPerature:BWIDth?
```
<Frequency> = 2 Hz, 8 Hz, 30 Hz

Example
```
:CHANNEL<ch>:TEMPERATURE:BWIDTH 2.0HZ
:CHANNEL<ch>:TEMPERATURE:BWIDTH?
-> 2.0E+00
```

Description An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

:CHANnel<ch>:TEMPerature:COUPling

Function Sets or queries input coupling when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:TEMPerature:
COUPling {TC|DC|GND}
:CHANnel<ch>TEMPerature:COUPling?
```

Example
```
:CHANNEL<ch>:TEMPerature:COUPlING DC
:CHANNEL<ch>:TEMPerature:COUPlING?
-> DC
```

Description An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

:CHANnel<ch>:TEMPerature:RJC

Function Sets or queries the RJC when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:TEMPerature:
RJC {<boolean>}
:CHANnel<ch>:TEMPerature:RJC?
```

Example
```
:CHANNEL<ch>:TEMPERATURE:RJC ON
:CHANNEL<ch>:TEMPERATURE:RJC? -> 1
```

Description An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

:CHANnel<ch>:TEMPerature:TYPE

Function Sets or queries the thermocouple type when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:TEMPerature:TYPE {K|E|J|T|
L|U|N|R|S|B|W|Au7fe}
:CHANnel<ch>:TEMPerature:TYPE?
```

Example
```
:CHANNEL<ch>:TEMPERATURE:TYPE K
:CHANNEL<ch>:TEMPERATURE:TYPE? -> K
```

Description An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

:CHANnel<ch>:TEMPerature:UNIT

Function Sets or queries the unit of measurement values when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>:TEMPerature:UNIT {C|F|K}
:CHANnel<ch>:TEMPerature:UNIT?
```

Example
```
:CHANNEL<ch>:TEMPERATURE:UNIT C
:CHANNEL<ch>:TEMPERATURE:UNIT? -> C
```

Description An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

:CHANnel<ch>[:VOLTage]:BWIDth

Function Sets or queries the bandwidth limit when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:BWIDth {FULL|
<Frequency>}
:CHANnel<ch>[:VOLTage]:BWIDth?
```

<Frequency> = 500 Hz, 5 kHz, 50 kHz, 500 kHz (When the module is 701250 or 701255)
400 Hz, 4 kHz, 40 kHz (When the module is 701251)
2 Hz, 8 Hz, 30 Hz (When the module is 701265)
100 Hz, 1 kHz, 10 kHz (When the module is 701260)
4 kHz, 400 Hz, 40 Hz (When the module is 701261 and coupling is not set to TC)
Auto, 4 kHz, 400 Hz, 40 Hz (When the module is 701262 and coupling is not set to TC)
2 MHz, 1.28 MHz, 640 kHz, 160 kHz, 80 kHz, 40 kHz, 20 kHz, 10 kHz (When the module is 720210)

Example
```
:CHANNEL<ch>:VOLTAGE:BWIDTH FULL
:CHANNEL<ch>:VOLTAGE:BWIDTH? -> FULL
```

Description An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:COUPling

Function Sets or queries input coupling when a voltage module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:COUPling {AC|DC|
GND|ACRMS|DCRMS|TC}
:CHANnel<ch>[:VOLTage]:COUPling?
```

Example
```
:CHANNEL<ch>:VOLTAGE:COUPlING DC
:CHANNEL<ch>:VOLTAGE:COUPlING?  -> DC
```

Description
- An error occurs if a voltage module is not installed.
- The following modules can be set to TC:
  701265/701261/701262
- The following module can be set to DCRMS or ACRMS
  70160
- The following module cannot be set to AC
  701265

:CHANnel<ch>[:VOLTage]:INVert

Function Sets or queries whether or not the display is inverted when a voltage module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:
INVert {<boolean>}
:CHANnel<ch>[:VOLTage]:INVert?
```

Example
```
:CHANNEL<ch>:VOLTAGE:INVERT ON
:CHANNEL<ch>:VOLTAGE:INVERT? -> 1
```

Description An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:LSCale:BVALue

Function Sets or queries linear scaling offset value B when a voltage module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:LSCale:
BVALue {<NRf>}
:CHANnel<ch>[:VOLTage]:LSCale:BVALue?
```
<NRf> = –9.9999E+30 to 9.9999E+30

Example
```
:CHANNEL<ch>:VOLTAGE:LSCALE:BVALUE 10
:CHANNEL<ch>:VOLTAGE:LSCALE:BVALUE?
-> 10.0000E+00
```

Description An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:LSCale:DISPlaytype:MODE

Function Sets or queries the display format for linear scaling.

Syntax
```
:CHANnel<ch>[:VOLTage]:LSCale:
DISPlaytype:MODE {EXPonent|FLOating}
:CHANnel<ch>[:VOLTage]:LSCale:
DISPlaytype:MODE?
```

Example
```
:CHANNEL<ch>:VOLTAGE:LSCALE:
DISPLAYTYPE:MODE EXPONENT
:CHANNEL<ch>:VOLTAGE:LSCALE:
DISPLAYTYPE:MODE? -> EXPONENT
```

:CHANnel<ch>[:VOLTage]:LSCale:DISPlaytype:DECimalnum

Function Sets or queries the decimal place when the display format for linear scaling is set to Floating.

Syntax
```
:CHANnel<ch>[:VOLTage]:LSCale:
DISPlaytype:DECimalnum {<NRf>|AUTO}
:CHANnel<ch>[:VOLTage]:LSCale:
DISPlaytype:DECimalnum?
```
<NRf> = 0 to 3

Example
```
:CHANNEL<ch>:VOLTAGE:LSCALE:
DISPLAYTYPE:DECIMALNUM AUTO
:CHANNEL<ch>:VOLTAGE:LSCALE:
DISPLAYTYPE:DECIMALNUM? -> AUTO
```

:CHANnel<ch>[:VOLTage]:LSCale:DISPlaytype:SUBunit

Function Sets or queries the sub unit when the display format for linear scaling is set to Floating.

Syntax
```
:CHANnel<ch>[:VOLTage]:LSCale:
DISPlaytype:SUBunit {AUTO|NONE|PICO|
NANO|MICRo|MILI|KILO|MEGA|GIGA|TERA}
:CHANnel<ch>[:VOLTage]:LSCale:
DISPlaytype:SUBunit?
```

Example
```
:CHANNEL<ch>:VOLTAGE:LSCALE:
DISPLAYTYPE:SUBUNIT AUTO
:CHANNEL<ch>:VOLTAGE:LSCALE:
DISPLAYTYPE:SUBUNIT? -> AUTO
```

:CHANnel<ch>[:VOLTage]:LSCale:GETMeasure

Function Measures the X values of P1 and P2 for linear scaling when a voltage module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:LSCale:
GETMeasure {P1X|P2X}
```

Example
```
:CHANNEL<ch>:VOLTAGE:GETMeasure P1X
```

Description An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:LSCale:MODE

Function Sets or queries linear scaling when a voltage module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:LSCale:
MODE {AXB|OFF|P12}
:CHANnel<ch>[:VOLTage]:LSCale:MODE?
```

Example
```
:CHANNEL<ch>:VOLTAGE:LSCALE:MODE AXB
:CHANNEL<ch>:VOLTAGE:LSCALE:MODE?
-> AXB
```

Description An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:LSCale:{P1X|P1Y|P2X|P2Y}

Function Sets or queries the X or Y value of P1 or P2 for linear scaling when a voltage module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:LSCale:{P1X|P1Y|P2X|P2Y} {<NRf>}
:CHANnel<ch>[:VOLTage]:LSCale:{P1X|P1Y|P2X|P2Y}?
```
For P1X and P2X, **<NRf>** = –9.9999E+30 to 9.9999E+30
For P1Y and P2Y, **<NRf>** = –9.9999E+25 to 9.9999E+25

Example
```
:CHANNEL<ch>:VOLTAGE:LSCALE:P1X 10
:CHANNEL<ch>:VOLTAGE:LSCALE:P1X?
-> 10.0000E+00
```

Description An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:LSCale:UNIT

Function Sets or queries the unit of measurement to attach to the result of linear scaling when a voltage module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:LSCale:UNIT {<String>}
:CHANnel<ch>[:VOLTage]:LSCale:UNIT?
```
<String> = Up to 4 characters

Example
```
:CHANNEL<ch>:VOLTAGE:LSCALE:UNIT "RPM"
:CHANNEL<ch>:VOLTAGE:LSCALE:UNIT?
-> "RPM"
```

Description An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:PROBe

Function Sets or queries the probe type when a voltage module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:PROBe {<NRf>|C10|C100}
:CHANnel<ch>[:VOLTage]:PROBe?
```
<NRf> = 1, 10, 100, 1000

Example
```
:CHANNEL<ch>:VOLTAGE:PROBE 10
:CHANNEL<ch>:VOLTAGE:PROBE? -> 10
```

Description
- An error occurs if a voltage module is not installed.
- Modules 701265, 701261, 701262 cannot be set or queried.
- The initial value is 10 (1 for module 701260).

:CHANnel<ch>[:VOLTage]:VDIV

Function Sets or queries the V/div value when a voltage module is installed in the specified channel (slot).

Syntax
```
:CHANnel<ch>[:VOLTage]:VDIV {<Voltage>|<Current>}
:CHANnel<ch>[:VOLTage]:VDIV?
```
<Voltage> = 5 mV to 20 V (When the probe attenuation is 1:1 on module 701250 and 701255)
1 mV to 20 V (When the probe attenuation is 1:1 on module 701251)
20 mV to 200 V (When the probe attenuation is 1:1 on module 701260)
0.1 mV to 10 V (When the module is 701265)
5 mV to 20 V (When the module is 701261 or 701262)
10 mV to 20 V (When the module is 720210)

Example
```
:CHANNEL<ch>:VOLTAGE:VDIV 5V
:CHANNEL<ch>:VOLTAGE:VDIV? -> 5.000E+00
```

Description An error occurs if a voltage module is not installed.

4.5 GONogo Group

The GONogo group deals with GO/NO-GO judgment. GONogo group commands are only valid when the measuring mode is Triggered mode.

:GONogo:ACONdition

Function Sets or queries the GO/NO-GO judgment action condition.

Syntax
```
:GONogo:ACONdition {ALWays|FAILure|SUCCess}
:GONogo:ACONdition?
```

Example
```
:GONOGO:ACONDITION FAILURE
:GONOGO:ACONDITION -> FAILURE
```

:GONogo:ACTion:BUZZer

Function Sets or queries whether or not a beep is sounded when the condition is met.

Syntax
```
:GONogo:ACTion:BUZZer {<boolean>}
:GONogo:ACTion:BUZZer?
```

Example
```
:GONOGO:ACTION:BUZZER OFF
:GONOGO:ACTION:BUZZER?-> 0
```

:GONogo:ACTion:MAIL:ADDRess

Function Sets or queries the destination e-mail address for when the condition is met.

Syntax
```
:GONogo:ACTion:MAIL:ADDRess {<String>}
:GONogo:ACTion:MAIL:ADDRess "yoko@yokogawa.jp.com"
:GONogo:ACTion:MAIL:ADDRess? -> "yoko@yokogawa.jp.com"
```

Description This command can be used when the optional Ethernet interface is installed.

:GONogo:ACTion:MAIL:COUNt

Function Sets or queries the e-mail transmission limit for when the condition is met.

Syntax
```
:GONogo:ACTion:MAIL:COUNt {<NRf>}
:GONogo:ACTion:MAIL:COUNt?
```
<NRf> = 1 to 100

Example
```
:GONOGO:ACTION:MAIL:COUNT 100
:GONOGO:ACTION:MAIL:COUNT?-> 100
```

Description This command can be used when the optional Ethernet interface is installed.

:GONogo:ACTion:MAIL:MODE

Function Sets or queries whether or not an e-mail is sent when the condition is met.

Syntax
```
:GONogo:ACTion:MAIL:MODE {<boolean>}
:GONogo:ACTion:MAIL:MODE?
```

Example
```
:GONOGO:ACTION:MAIL:MODE OFF
:GONOGO:ACTION:MAIL:MODE?-> 0
```

Description This command can be used when the optional Ethernet interface is installed.

:GONogo:ACTion:SAVE[:MODE]

Function Sets or queries whether or not waveform data is saved to the storage media when the condition is met.

Syntax
```
:GONogo:ACTion:SAVE[:MODE] {<boolean>}
:GONogo:ACTion:SAVE[:MODE]?
```

Example
```
:GONOGO:ACTION:SAVE:MODE OFF
:GONOGO:ACTION:SAVE:MODE? -> 0
```

Description Set or query the media type by using the :FILE:DIRECTORY:DRIVE command.

:GONogo:ACTion:SAVE:TYPE

Function Sets or queries the data type for saving waveform data to the storage media when the condition is met.

Syntax
```
:GONogo:ACTion:SAVE:TYPE {ASCii|BINary|FLOat}
:GONogo:ACTion:SAVE:TYPE?
```

Example
```
:GONOGO:ACTION:SAVE:TYPE ASCII
:GONOGO:ACTION:SAVE:TYPE? -> ASCII
```

:GONogo:AREA

Function Sets or queries the waveform area that is judged.

Syntax
```
:GONogo:AREA {CURSor|FULL}
:GONogo:AREA?
```

Example
```
:GONOGO:AREA FULL
:GONOGO:AREA? -> FULL
```

:GONogo:COUNt?

Function Queries the number of performed GO/NO-GO judgments.

Syntax
```
:GONogo:COUNt?
```

Example
```
:GONOGO:COUNT? -> 10
```

:GONogo:LOGic

Function Sets or queries the GO/NO-GO logical condition.

Syntax
```
:GONogo:LOGic {AND|OR}
:GONogo:LOGic?
```

Example
```
:GONOGO:LOGIC AND
:GONOGO:LOGIC? -> AND
```

:GONogo:MODE

Function Sets or queries the GO/NO-GO judgment mode.

Syntax
```
:GONogo:MODE {OFF|PARameter}
:GONogo:MODE?
```

Example
```
:GONOGO:MODE PARAMETER
:GONOGO:MODE? -> PARAMETER
```

:GONogo:NGCount?

Function Queries the GO/NO-GO judgment NO-GO count.
Syntax :GONogo:NGCount?
Example :GONOGO:NGCOUNT? -> 10

:GONogo:PARameter:ITEM<x>:CAUSe?

Function Queries whether or not the specified waveform parameter is the cause of a NO-GO judgment.
Syntax :GONogo:PARameter:ITEM<x>:CAUSe?
The **<x>** in ITEM**<x>** = 1 to 16
Example :GONOGO:PARAMETER:ITEM1:CAUSE? -> 1
Description When the waveform parameter is the cause of a NO-GO judgment, the command returns 1. Otherwise, the command returns 0.

:GONogo:PARameter:ITEM<x>:MODE

Function Sets or queries whether or not the specified waveform parameter is OFF, or what its judgment criterion is.
Syntax :GONogo:PARameter:ITEM<x>:MODE {OFF|IN|OUT}
:GONogo:PARameter:ITEM<x>:MODE?
The **<x>** in ITEM**<x>** = 1 to 16
Example :GONOGO:PARAMETER:ITEM1:MODE IN
:GONOGO:PARAMETER:ITEM1:MODE? -> IN

:GONogo:PARameter:ITEM<x>:PARam?

Function Queries the waveform parameter of the specified judgment condition.
Syntax :GONogo:PARameter:ITEM<x>:PARam?
Example :GONOGO:PARAMETER:ITEM1:PARAM? -> AVER

:GONogo:PARameter:ITEM<x>:TRACe

Function Sets or queries the channel number of the specified judgment condition.
Syntax :GONogo:PARameter:ITEM<x>:TRACe {<NRf>}
:GONogo:PARameter:ITEM<x>:TRACe?
The **<x>** in ITEM**<x>** = 1 to 16
<NRf> = 1 to 1024
Example :GONOGO:PARAMETER:ITEM1:TRACE 1
:GONOGO:PARAMETER:ITEM1:TRACE? -> 1

:GONogo:PARameter:ITEM<x>:TYPE:<Parameter>

Function Sets or queries the upper and lower limits of the judgment area for the specified judgment condition.
Syntax :GONogo:PARameter:ITEM<x>:TYPE:<Parameter>{<{Voltage|DONTcare}>,<{Voltage|DONTcare}>|<{Current|DONTcare},<{Current|DONTcare}>|<{Time|DONTcare},<{Time|DONTcare}>|<{Frequency|DONTcare},<{Frequency|DONTcare}>|<{<NRf>|DONTcare}>,<{<NRf>|DONTcare}>}
:GONogo:PARameter:ITEM<x>:TYPE:<Parameter>?
The **<x>** in ITEM**<x>** = 1 to 16
<Parameter> = {AMPLitude|AVERage|AVGFreq|AVGPeriod|BWIDth1|BWIDth2|DELay|DUTYcycle|FALL|FREQuency|HIGH|LOW|MAXimum|MIDDle|MINimum|NOVershoot|NWIDth|PERiod|PNUMber|POVershoot|PTOPeak|PWIDth|RISE|RMS|SDEViation|TY1Integ|TY2Integ|XY1Integ|XY2Integ}
<Voltage>, **<Current>**, **<Time>**, **<Frequency>**, **<NRf>** = See the *SL1000 High Speed Data Acquisition Unit User's Manual* for details.
Example :GONOGO:PARAMETER:ITEM1:TYPE:AVERAGE 100MV,-100MV
:GONOGO:PARAMETER:ITEM1:TYPE:AVERAGE?
-> 100.000E-03,-100.000E-03
Description Only the values of parameters that have been set with this command can be queried.

:GONogo:PARameter:ITEM<x>:VALue?

Function Queries the automated measurement value of the specified GO/NO-GO judgment parameter.
Syntax :GONogo:PARameter:ITEM<x>:VALue?
The **<x>** in ITEM**<x>** = 1 to 16
Example :GONOGO:PARAMETER:ITEM1:VALUE?
-> 50.000000E+00
Description When the mode is set to OFF or when the value is otherwise immeasurable, the command returns "NaN" (not a number).

:GONogo:REMote

Function Sets or queries the GO/NO-GO judgment remote mode.
Syntax :GONogo:REMote {<boolean>}
:GONogo:REMote?
Example :GONOGO:REMOTE ON
:GONOGO:REMOTE? -> 1

:GONogo:RSTatus?

Function Queries the most recent GO/NO-GO judgment.
Syntax GONogo:RSTatus?

Example GONogo:RSTatus? -> 0

Description The command returns 0 when the judgment is GO and it returns 1 when the judgment is NO-GO.

:GONogo:SEQuence

Function Sets or queries the GO/NO-GO judgment action mode.

Syntax :GONogo:SEQuence {CONTinue|SINGle}
:GONogo:SEQuence?

Example :GONOGO:SEQUENCE CONTINUE
:GONOGO:SEQUENCE? -> CONTINUE

:GONogo:STOPcount

Function Sets or queries the GO/NO-GO judgment count.

Syntax :GONogo:STOPcount {<NRf>|INFinity}
:GONogo:STOPcount?
<NRf> = 1 to 65536

Example :GONOGO:STOPCOUNT 10
:GONOGO:STOPCOUNT? -> 10

4.6 MEASure Group

The MEASure group deals with the automated measurement of waveform parameters. MEASure group commands are only valid when the measuring mode is Triggered mode.

:MEASure:AREA

Function Sets or queries the automatically measured waveform area for the waveform parameters.

Syntax
```
MEASure:AREA {CURSor|FULL}
MEASure:AREA?
```

Description When FULL is specified, the entire history waveform memory is subject to computation. When CURSor is specified, only the range set with MEASsire:CRANge is subject to computation.

:MEASure:CHANnel<ch>:DPRoximal:MODE

Function Sets or queries the distal, mesial, and proximal point mode setting.

Syntax
```
:MEASure:CHANnel<ch>:DPRoximal:
MODE {PERCent|UNIT}
:MEASure:CHANnel<ch>:DPRoximal:MODE?
```

Example
```
:MEASURE:CHANNEL1:DPROXIMAL:
MODE PERCENT
:MEASURE:CHANNEL1:DPROXIMAL:MODE?
-> PERCENT
```

:MEASure:CHANnel<ch>:DPRoximal:PERCent

Function Sets or queries the distal, mesial, and proximal points as percentages.

Syntax
```
:MEASure:CHANnel<ch>:DPRoximal:
PERCent {<NRf>,<NRf>,<NRf>}
:MEASure:CHANnel<ch>:DPRoximal:
PERCent?
```
<NRf> = 0 to 100 (%, in 0.1% steps)
The values specify the proximal, mesial, and distal points in that order.

Example
```
:MEASURE:CHANNEL1:DPROXIMAL:
PERCENT 40,60,80
:MEASURE:CHANNEL1:DPROXIMAL:PERCENT?
-> 40.0,60.0,80.0
```

:MEASure:CHANnel<ch>:DPRoximal:UNIT

Function Sets or queries the distal, mesial, and proximal points.

Syntax
```
:MEASure:CHANnel<ch>:DPRoximal:
UNIT {<Voltage>,<Voltage>,<Voltage>|<Cu
rrent>,<Current>,<Current>|<NRf>,<NRf>
,<NRf>}
:MEASure:CHANnel<ch>:DPRoximal:UNIT?
```
For CHANnel**<ch>**: The settable ranges of **<Voltage>**, **<Current>**, and **<NRf>** vary depending on the range and offset settings. For details, see the *SL1000 High Speed Data Acquisition Unit User's Manual*.
The values specify the proximal, mesial, and distal points in that order.

Example
```
:MEASURE:CHANNEL1:DPROXIMAL:
UNIT -50V,0V,50V
:MEASURE:CHANNEL1:DPROXIMAL:UNIT?
-> -50.000E+00,0.0E+00,50.000E+00
```

:MEASure:CHANnel<ch>:METHod

Function Sets or queries the high/low point setting method.

Syntax
```
:MEASure:CHANnel<ch>:METHod {AUTO|
MAXMin}
:MEASure:CHANnel<ch>:METHod?
```

Example
```
:MEASURE:CHANNEL1:METHOD AUTO
:MEASURE:CHANNEL1:METHOD? -> AUTO
```

:MEASure:CHANnel<ch>:<Parameter>:STATe

Function Sets or queries the on/off state of the measurement of the specified waveform parameter.

Syntax
```
:MEASure:CHANnel<ch>:<Parameter>:
STATe {<boolean>}
:MEASure:CHANnel<ch>:<Parameter>:
STATe?
<Parameter> = {AMPLitude|AVERage|
AVGFreq|AVGPeriod|BWIDth1|BWIDth2|
DUTYcycle|FALL|FREQuency|HIGH|LOW|
MAXimum|MIDDle|MINimum|NOVershoot|
NWIDth|PERiod|PNUMber|POVershoot|
PTOPeak|PWIDth|RISE|RMS|SDEViation|
TY1Integ|TY2Integ|XY1Integ|XY2Integ}
```

Example
```
:MEASURE:CHANNEL1:AVERAGE:STATE ON
:MEASURE:CHANNEL1:AVERAGE:STATE? -> 1
```

:MEASure:CHANnel<ch>:<Parameter>:VALue?

Function Queries the automated measurement value of the specified waveform parameter.

Syntax `:MEASure:CHANnel<ch>:<Parameter>:VALue? {<NRf>}`
<NRf> = 1 to 48000
`<Parameter>` = {AMPLitude|AVERage|AVGFreq|AVGPeriod|BWIDth1|BWIDth2|DUTYcycle|FALL|FREQuency|HIGH|LOW|MAXimum|MIDDle|MINimum|NOVershoot|NWIDth|PERiod|PNUMber|POVershoot|PTOPeak|PWIDth|RISE|RMS|SDEViation|TY1Integ|TY2Integ|XY1Integ|XY2Integ}

Example `:MEASURE:CHANNEL1:AVERAGE:VALUE? -> 115.95507E-03`

Description If the value is immeasurable, the command returns "NaN" (not a number). The "**<NRf>**" at the end is used to specify what parameter value to query. It indicates the order of the parameter value since statistical processing began. If the specified parameter does not exist, the command returns "NaN" (not a number).
<NRf> can be omitted. If a value is not entered for **<NRf>**, the most recent waveform parameter value in the memory will be queried. When a value is entered for **<NRf>**, the order starts with the newest waveform in the memory and increases as the waveform parameter values get older.

:MEASure:CRANge

Function Sets or queries the waveform parameter measurement range.

Syntax `:MEASure:CRANge {<NRf>,<NRf>}`
`:MEASure:CRANge?`
<NRf> = 0 to 134217728

:MEASure:MODE

Function Sets or queries the waveform parameter automated measuring mode.

Syntax `:MEASure:MODE {OFF|ON}`
`:MEASure:MODE?`

Example `:MEASURE:MODE ON`
`:MEASURE:MODE? -> ON`

:MEASure:WAIT?

Function Waits for the execution of waveform parameter automated measurement with a set timeout.

Syntax `:MEASure:WAIT? {<NRf>}`
<NRf> = 1 to 36000 (the timeout specified in 100 ms intervals)

Example `:MEASURE:WAIT? 100 -> 1`

Description
- The command returns 0 if the automated measurement finishes within the specified timeout. If automated measurement does not finish, or if it was never taking place to begin with, the command returns 1.
- Even if you set a long timeout, the command will return 0 as soon as automated measurement finishes.

4.7 TRIGger Group

TRIGger group commands are only valid when the measuring mode is Triggered mode.

:TRIGger:COMBination:CHANnel<ch>:COMBination

Function Sets or queries the trigger AND/OR state of all of the bits in the specified channel for the combination trigger class.

Syntax
```
:TRIGger:COMBination:CHANnel<ch>:
COMBination {AND|OR}
:TRIGger:COMBination:CHANnel<ch>:
COMBination?
```

Example
```
:TRIGGER:COMBINATION:CHANNEL<ch>:
COMBINATION AND
:TRIGGER:COMBINATION:CHANNEL<ch>:
COMBINATION? -> AND
```

:TRIGger:COMBination:CHANnel<ch>:HYSTeresis<x>

Function Sets or queries the trigger hysteresis of the specified channel in the combination trigger class.

Syntax
```
:TRIGger:COMBination:CHANnel<ch>:
HYSTeresis<x> {HIGH|LOW|MIDDLE}
:TRIGger:COMBination:CHANnel<ch>:
HYSTeresis<x>?
```
The **<x>** in HYSTeresis**<x>** = 1, 2
If TYPE is RISE|FALL|HIGH|LOW|BISLope, only 1 is used.
If TYPE is WLIn|WLOut|WINIn|WINOut, both 1 and 2 are used.
1 is the upper limit. 2 is the lower limit.

Example
```
:TRIGGER:COMBINATION:CHANNEL<ch>:
HYSTERESIS2 HIGH
:TRIGGER:COMBINATION:CHANNEL<ch>:
HYSTERESIS? -> HIGH
```

:TRIGger:COMBination:CHANnel<ch>:LEVel<x>

Function Sets or queries the trigger level of the specified channel in the combination trigger class.

Syntax
```
:TRIGger:COMBination:CHANnel<ch>:
LEVel<x> {<Voltage>|<NRf>|<Current>}
:TRIGger:COMBination:CHANnel<ch>:
LEVel<x>?
```
The **<x>** in LEVel**<x>** = 1, 2
If TYPE is RISE|FALL|HIGH|LOW|BISLope, only 1 is used.
If TYPE is WLIn|WLOut|WINIn|WINOut, both 1 and 2 are used.
1 is the upper limit. 2 is the lower limit.

Example
```
:TRIGGER:COMBINATION:CHANNEL<ch>:
LEVEL1 0V
:TRIGGER:COMBINATION:CHANNEL<ch>:
LEVEL1? -> 0.0E+00
```

Description For information about the settable range, see "TRIGger[:SIMPle]:LEVel."

:TRIGger:COMBination:CHANnel<ch>:TYPE

Function Sets or queries the trigger type of the specified channel in the combination trigger class.

Syntax
```
:TRIGger:COMBination:CHANnel<ch>:
TYPE {OFF|RISE|FALL|HIGH|LOW|BISLope|
WLIn|WLOut|WINIn|WINOut}
:TRIGger:COMBination:CHANnel<ch>:TYPE?
```

Example
```
:TRIGGER:COMBINATION:CHANNEL<ch>:
TYPE LOW
:TRIGGER:COMBINATION:CHANNEL<ch>:TYPE?
-> LOW
```

:TRIGger:COMBination:EXTernal:TYPE

Function Sets or queries the external trigger type of the specified channel in the combination trigger class.

Syntax
```
:TRIGger:COMBination:EXTernal:
TYPE {OFF|RISE|FALL|HIGH|LOW}
:TRIGger:COMBination:EXTernal:TYPE?
```

Example
```
:TRIGGER:COMBINATION:EXTERNAL:TYPE RISE
:TRIGGER:COMBINATION:EXTERNAL:TYPE?
-> RISE
```

:TRIGger:COMBination:MODE

Function Sets or queries the combination mode of the combination trigger class.

Syntax
```
:TRIGger:COMBination:MODE {AND|OR}
:TRIGger:COMBination:MODE?
```

Example
```
:TRIGGER:COMBINATION:EXTERNAL:MODE OR
:TRIGGER:COMBINATION:EXTERNAL:MODE?
-> OR
```

:TRIGger:DELay

Function Sets or queries the delay (the time between the trigger point and the trigger position).

Syntax `:TRIGger:DELay {<Time>}`
`:TRIGger:DELay?`
<Time> = 0 to 10 s

Example `:TRIGGER:DELAY 2US`
`:TRIGGER:DELAY? -> 2.00000E-06`

Description The resolution depends on the sample rate. (The resolution is 0.1 divided by the sample rate.) However, the highest resolution that can be set is 10 nanoseconds (if the sample rate is higher than 10 MS/s, the resolution will be 10 nanoseconds). The value is fixed at 0 when an external clock is used.

:TRIGger:HOLDoff:TIME

Function Sets or queries the trigger hold off time.

Syntax `:TRIGger:HOLDoff:TIME {<Time>}`
`:TRIGger:HOLDoff:TIME?`
<Time> = 0 to 10 s (with a resolution of 10 nanoseconds)

Example `:TRIGGER:HOLDOFF:TIME 500NS`
`:TRIGGER:HOLDOFF:TIME? -> 500.000E-09`

:TRIGger[:SIMPle]:HYSTeresis

Function Sets or queries the hysteresis of the simple trigger.

Syntax `:TRIGger[:SIMPle]:HYSTeresis {HIGH|LOW|MIDDle}`
`:TRIGger[:SIMPle]:HYSTeresis?`

Example `TRIGGER:SIMPLE:HYSTERESIS MIDDLE`
`:TRIGGER:SIMPLE:HYSTERESIS? -> MIDDLE`

Description The hysteresis cannot be set or queried when the trigger source is set to EXTernal, LINE, or TIME.

:TRIGger[:SIMPle]:LEVel

Function Sets or queries the simple trigger of the channel specified by :TRIGger[:SIMPle]:SOURce.

Syntax `:TRIGger[:SIMPle]:LEVel {<Voltage>|<NRf>|<Current>}`
`:TRIGger[:SIMPle]: LEVel?`

Example `:TRIGGER:SIMPLE:LEVEL 0V`
`:TRIGGER:SIMPLE:LEVEL? -> 0.0E+00`

Description The hysteresis cannot be set or queried when the trigger source is set to EXTernal, LINE, or TIME. For the settable ranges, see the table below.

Trigger level settable ranges and resolutions:

Input	Settable Range	Resolution
Voltage	± (V/div) × 10	1/100 of the V/div value Example: 1V/div = 0.01 V resolution 500mV/div = 0.005V 200 mV/div = 0.002 V
Temperature	The measurement range of each type of thermocouple.	0.1 (For C, K, and F)
Strain	±(Measurable range)	1 μSTR 0.0005 mV/V
Acceleration	×0.1 ± 100000 unit ×0.2 ± 50000 unit ×0.5 ± 20000 unit ×1 ± 10000 unit ×2 ± 5000 unit ×5 ± 2000 unit ×10 ± 1000 unit ×20 ± 500 unit ×50 ± 200 unit ×100 ± 100 unit	0.01 unit
Frequency (701280)	± (V/div) × 10	Equivalent to 0.0005 div or 0.001 div. The smallest value is 0.001 Hz. Example: 1 Hz/div = 0.001 Hz resolution 2 Hz/div = 0.002 Hz 5 Hz/div = 0.005 Hz 10 Hz/div = 0.005 Hz
RPM (701280)	± (V/div) × 10	Equivalent to 0.0005 div or 0.001 div. Example: 1000 rpm/div = 0.5 rpm resolution 2000 rpm/div = 2 rpm 5000 rpm/div = 5 rpm

Input	Settable Range	Resolution
RPS (701280)	± (V/div) × 10	Equivalent to 0.0005 div or 0.001 div. Example: 1 rps/div = 0.5 mrps resolution 2 rps/div = 0.002 rps 5 rps/div = 0.005 rps
Interval (701280)	± (V/div) × 10	Equivalent to 0.0005 div or 0.001 div. Example: 1 ms/div = 0.5 μs 2 ms/div = 2 μs 5 ms/div = 5 μs
Duty (701280)	± (V/div) × 10	Equivalent to 0.001 div 1%/div = 0.001% 2%/div = 0.002% 5%/div = 0.005%
Power source frequency (701280)	Central frequency ± (V/div) × 10	2 Hz/div = 0.002 Hz 1 Hz/div = 0.001 Hz 0.5 Hz/div = 0.001 Hz 0.2 Hz/div = 0.001 Hz 0.1 Hz/div = 0.001 Hz
Pulse width (701280)	± (V/div) × 10	Equivalent to 0.0005 div or 0.001 div. Example 1 ms/div = 0.5 μs 2 ms/div = 2 μs 5 ms/div = 5 μs
Integration (701280)	± (V/div) × 10	The float setting
Speed (701280)	± (V/div) × 10	The float setting

:TRIGger[:SIMPle]:SLOPe

Function Sets or queries the simple trigger type of the channel specified by :TRIGger[:SIMPle]:SOURce.

Syntax `:TRIGger[:SIMPle]:SLOPe {OFF|RISE|FALL|BISLope}`
`:TRIGger[:SIMPle]:SLOPe?`

Example `:TRIGGER:SIMPLE:SLOPE RISE`
`:TRIGGER:SIMPLE:SLOPE? -> RISE`

Description The hysteresis cannot be set or queried when the trigger source is set to EXTernal, LINE, or TIME.

:TRIGger:SIMPle:SOURce

Function Sets or queries the simple trigger source.

Syntax `:TRIGger:SIMPle:SOURce {<NRf>|EXTernal|LINE|TIME}`
`:TRIGger:SIMPle:SOURce?`

Example `:TRIGGER:SIMPLE:SOURCE 1`
`:TRIGGER:SIMPLE:SOURCE? -> 1`

Description Of the :TRIGger:SIMPle commands, only the :TRIGger:SIMPle:SOURce command requires that ":SIMPle" not be omitted.

:TRIGger:TIMer:DATE

Function Sets or queries the date of the time trigger.

Syntax `:TRIGger:TIMer:DATE <String>`
`:TRIGger:TIMer:DATE?`
<String> = YYYY/MM/DD
Year can be set in the range of 2000 to 2099.

Example `TRIGGER:TIMER:DATE "2007/12/04"`
`:TRIGGER:TIMER:DATE? -> "2007/12/04"`

:TRIGger:TIMer:INTerval

Function Sets or queries the trigger interval of the time trigger.

Syntax `:TRIGger:TIMer:INTerval {MIN1|MIN2|MIN3|MIN4|MIN5|MIN6|MIN7|MIN8|MIN9|MIN10|MIN15|MIN20|MIN9|MIN10|MIN15|MIN20|MIN25|MIN30|MIN40|MIN45|MIN50|HOUR1|HOUR2|HOUR3|HOUR4|HOUR5|HOUR6|HOUR7|HOUR8|HOUR9|HOUR10|HOUR11|HOUR12|HOUR18|HOUR24}`
`:TRIGger:TIMer:INTerval?`

Example `:TRIGGER:TIMER:INTERVAL HOUR1`
`:TRIGGER:TIMER:INTERVAL? -> HOUR1`

:TRIGger:TIMer:TIME

Function Sets or queries the time of the time trigger.

Syntax `:TRIGger:TIMer:TIME <String>`
`:TRIGger:TIMer:TIME?`
<String> = HH:MM:SS

Example `:TRIGGER:TIMER:TIME "12:34:56"`
`:TRIGGER:TIMER:TIME? -> "12:34:56"`

:TRIGger:TYPE

Function Sets or queries the trigger type.

Syntax `:TRIGger:TYP {COMBination|SIMPle}`
`:TRIGger:TYPE?`

Example `:TRIGGER:TYPE SIMPLE`
`:TRIGGER:TYPE? -> SIMPLE`

5.1 Library Errors

SxAPI functions return two kinds of errors: unit errors and library errors.

Error Codes	Description
1 to 9999	Unit errors These errors are returned from the SL1000. For details, see section 5.2.
10000 and greater	Library errors These errors are returned from SxAPI. For details, see section 3.19 and the descriptions below.

Library errors

Error Code	Description
0	No error, closed properly
10001	Timeout
10002	Cannot find the target unit
10003	Open failed
10004	Not opened
10005	Already opened
10006	Environment error
10007	Invalid parameter
10008	Send error
10009	Receive error
10010	The received data is not in block format.
10011	System error
10012	ID violation
10013	Communication command error
10014	Insufficient buffer
10016	Cannot find the target unit group
10017	Invalid unit group
10018	Handle type violation
10019	Handle error
10020	Cannot find handle
10021	Command string violation
10022	Data outside of the range has been specified.
10023	The specified data does not exist.
10024	Conflict error
10031	Internal error

5.2 Unit Errors

These errors are returned from the SL1000.

Communication Syntax Errors (100 to 199)

Error Code	Description
102	Syntax error. A syntax error other than one of the ones listed below.
103	Invalid separator. Separate parameters with a comma.
104	Data type error. Use the correct data type for each parameter.
105	GET not allowed. GET is not supported as a response to an interface message.
108	Parameter not allowed. Check the number of parameters.
109	Missing parameter. Be sure to include all necessary parameters.
111	Header separator error. Separate commands from parameters with a space.
112	Header separator error. Check the command length.
113	Undefined header. Confirm the command name.
114	Header suffix out of range. Check the command.
120	Numeric data error. A value must be specified where the syntax contains <NRf>.
123	Exponent too large. Where the syntax contains <NR3>, make the exponent that follows E smaller.
124	Too many digits. Limit numeric values to 255 digits or less.
128	Numeric data not allowed. Use a data type other than <NRf>.
131	Invalid suffix. Check the units where the syntax contains <Voltage>, <Time>, or <Frequency>.
134	Suffix too long. Check the units where the syntax contains <Voltage>, <Time>, or <Frequency>.
138	Suffix not allowed. Units of measurement can only be used where the syntax contains <Voltage>, <Time>, or <Frequency>.
141	Invalid character data. Be sure to select one of the listed choices when the syntax contains {...|...|...}.
144	Character data too long. Check the spelling of strings where the syntax contains {...|...|...}.
148	Character data not allowed. Use a data type other than {...|...|...}.
150	String data error. Enclose parameters with single or double quotation marks where the syntax contains <String>.
151	Invalid string data. The parameter is either too long, or it contains an unusable character.
158	String data not allowed. Use a data type other than <String>.
161	Invalid block data. <Block data> cannot be used.
168	Block data not allowed. <Block data> cannot be used.
171	Invalid expression. Mathematical operations cannot be used.
178	Expression data not allowed. Mathematical operations cannot be used.
181	Invalid outside macro definition. Does not conform to the IEEE488.2 macro specifications.

Communication Execution Errors (200 to 299)

Error Code	Description
221	Setting conflict. Check settings that are related to each other.
222	Data out of range. Check the ranges of the settings.
223	Too much data. Check data byte lengths.
224	Illegal parameter value. Check the ranges of the settings.
241	Hardware missing. Check that the specified options are all installed.
260	Expression error. Mathematical operations cannot be used.
270	Macro error. Does not conform to the IEEE488.2 macro specifications.
272	Macro execution error. Does not conform to the IEEE488.2 macro specifications.
273	Illegal macro label. Does not conform to the IEEE488.2 macro specifications.
275	Macro definition too long. Does not conform to the IEEE488.2 macro specifications.
276	Macro recursion error. Does not conform to the IEEE488.2 macro specifications.
277	Macro redefinition not allowed. Does not conform to the IEEE488.2 macro specifications.
278	Macro header not found. Does not conform to the IEEE488.2 macro specifications.

Communication Query Errors (400 to 499)

Error Code	Description
410	Query INTERRUPTED. Check the transmission and reception order.
420	Query UNTERMINATED. Check the transmission and reception order.
430	Query DEADLOCKED. Transmission will be stopped.
440	Query UNTERMINATED after indefinite response.

System Communication Error (912)

Error Code	Description
912	Fatal error in the communication driver. Contact customer service.

Appendix 1 Sample Programs

Visual Basic 6.0

Initializing and Setting Measurement Conditions

Connecting and Opening

The following code connects through USB to the unit group whose ID = 0, and returns the configurations of the units in the group to their factory default settings. It also specifies the reception of a measurement end event.

```
' Connect through USB
  Ret = SxInit(SxEvent1.hWnd, SX_WIRE_USB, "", hComm)

' Open the unit group whose group ID is 0.
  Ret = SxOpenGroup(hComm, 0, hGrp)

' Return unit configurations to their factory default
' settings.
  Ret = SxInitSetup(hGrp)

' Specify reception of a measurement end event.
  Ret = SxCreateEvent(hGrp, 0, SX_EV_ACQ_STOP)
```

Setting Measurement Conditions

The following code sets the measurement conditions as follows:

Acquisition channels = CH1, CH2, and CH3
All probes = “1:1”
All channel measuring ranges = ± 10[V]
Measuring mode = “Triggered”
Trigger mode = “Single”
Sampling rate = 10 k [S/s]
Measuring time = 0.1 s
Trigger position = 10%
Trigger source = CH1
Trigger level = 2.5 V

```
'     Turn CH1 to 3 measurement on and set the probe and range.
  For i = 0 to 2
'     Measurement = On
       Ret = SxSetAcqSwitch(SxChHndl(hGrp, i), 1)
'     Probe = "1:1"
       Ret = SxSetControl(SxChHndl(hGrp, i), ":CHAN<ch>:PROB 1")
'     Range = 10 V
       Ret = SxSetControl(SxChHndl(hGrp, i), ":CHAN<ch>:VDIV 1")
'     (The value is set to 1/10 of the range because the range is
'     specified using VDIV.)
  Next i

' Measuring mode = "Triggered"
  Ret = SxSetAcqMode(hGrp, SX_ACQ_TRIG)

' Trigger mode = "Single"
  Ret = SxSetTrigMode(hGrp, SX_TRIG_SINGLE)

' Sample interval = 100 us (= 10 kS/s)
  Ret = SxSetSamplingInterval(hGrp, 0.0001)

' Record length = 0.1 s
  Ret = SxSetAcqSpan(hGrp, 0.1)
```

```
' Trigger position = 10%
  Ret = SxSetTrigPos(hGrp, 10.0)

' Trigger source = CH1
  Ret = SxSetControl(hGrp, ":TRIG:SIMP:SOUR CH1")

' Trigger level = 2.5 V
  Ret = SxSetControl(hGrp, ":TRIG:SIMP:LEV 2.5")
```

Starting Measurement

The code below sends a command to the units to start measuring.

```
' Start measurement
  Ret = SxAcqStart(hGrp)
```

Saving Waveform Data

The code below saves the acquired waveform data to a (WDF format) file.

```
' Save the most recent waveform data
  Ret = SxSaveAcqData(SxUnitHndl(hGrp, 0), -1)
' Caution: -1 indicates the most recent data.
```

Closing

Closing and Disconnecting

The code below deletes the event handle, closes the unit group, and disconnects.

```
' Delete the event handle
  Ret = SxDeleteEvent(hGrp)

' Close the unit group
  Ret = SxCloseGroup(hGrp)

' Disconnect
  Ret = SxExit(hComm)
```

Visual Basic 2005

Initializing and Setting Measurement Conditions

Connecting and Opening

The following code connects through USB to the unit group whose ID = 0, and returns the configurations of the units in the group to their factory default settings. It also specifies the reception of a measurement end event.

```
' Connect through USB
  Ret = SxAPI1.Init(WIRE.USB, "", hComm)

' Open the unit group whose group ID is 0.
  Ret = SxAPI1.OpenGroup(hComm, 0, hGrp)

' Return unit configurations to their factory default settings.
  Ret = SxAPI1.InitSetup(hGrp)

' Specify reception of a measurement end event.
  Ret = SxAPI1.CreateEvent(hGrp, EV.ACQ_STOP)
```

Setting Measurement Conditions

The following code sets the measurement conditions as follows:

Acquisition channels = CH1, CH2, and CH3

All probes = “1:1”
All channel measuring ranges = ± 10[V]
Measuring mode = “Triggered”
Trigger mode = “Single”
Sampling rate = 10 k [S/s]
Measuring time = 0.1 s
Trigger position = 10%
Trigger source = CH1
Trigger level = 2.5 V

```
' Turn CH1 to 3 measurement on and set the probe and range.
  For i = 0 to 2
' Measurement = On
  Ret = SxAPI1.SetAcqSwitch(SxAPI1.ChHndl(hGrp, i), 1)
' Probe = "1:1"
  Ret = SxAPI1.SetControl(SxAPI1.ChHndl(hGrp, i), ":CHAN<ch>:PROB 1")
' Range = 10 V
  Ret = SxAPI1.SetControl(SxAPI1.ChHndl(hGrp, i), ":CHAN<ch>:VDIV 1")
' (The value is set to 1/10 of the range because the range is specified
' using VDIV.)
  Next i

' Measuring mode = "Triggered"
  Ret = SxAPI1.SetAcqMode(hGrp, ACQMODE.TRIG)

' Trigger mode = "Single"
  Ret = SxAPI1.SetTrigMode(hGrp, TRIGMODE.SINGLE)

' Sample interval = 100 us (= 10 kS/s)
  Ret = SxAPI1.SetSamplingInterval(hGrp, 0.0001)

' Record length = 0.1 s
  Ret = SxAPI1.SetAcqSpan(hGrp, 0.1)

' Trigger position = 10%
  Ret = SxAPI1.SetTrigPos(hGrp, 10.0)
```

```
' Trigger source = CH1
  Ret = SxAPI1.SetControl(hGrp, ":TRIG:SIMP:SOUR CH1")

' Trigger level = 2.5 V
  Ret = SxAPI1.SetControl(hGrp, ":TRIG:SIMP:LEV 2.5")
```

Starting Measurement

The code below sends a command to the units to start measuring.

```
' Start measurement
  Ret = SxAPI1.AcqStart(hGrp)
```

Saving Waveform Data

The code below saves the acquired waveform data to a (WDF format) file.

```
' Save the most recent waveform data
  Ret = SxAPI1.SaveAcqData(SxAPI1.UnitHndl(hGrp, 0), -1, "C:\Data\
  filename")
'    Caution: -1 indicates the most recent data.
```

Closing

Closing and Disconnecting

The code below deletes the event handle, closes the unit group, and disconnects.

```
' Delete the event handle
  Ret = SxAPI1.DeleteEvent(hGrp)

' Close the unit group
  Ret = SxAPI1.CloseGroup(hGrp)

' Disconnect
  Ret = SxAPI1.Exit(hComm)
```

Visual C#

Initializing and Setting Measurement Conditions

Connecting and Opening

The following code connects through USB to the unit group whose ID = 0, and returns the configurations of the units in the group to their factory default settings. It also specifies the reception of a measurement end event.

```
' Connect through USB
  Ret = sxAPI1.Init(WIRE.USB, "", ref hComm);

' Open the unit group whose group ID is 0.
  Ret = sxAPI1.OpenGroup(hComm, 0, ref hGrp);

' Return unit configurations to their factory default settings.
  Ret = sxAPI1.InitSetup(hGrp);

' Specify reception of a measurement end event.
  Ret = sxAPI1.CreateEvent(hGrp, (uint)EV.ACQ_STOP);
```

Setting Measurement Conditions

The following code sets the measurement conditions as follows:

Acquisition channels = CH1, CH2, and CH3
All probes = “1:1”
All channel measuring ranges = ± 10[V]
Measuring mode = “Triggered”
Trigger mode = “Single”
Sampling rate = 10 k [S/s]
Measuring time = 0.1 s
Trigger position = 10%
Trigger source = CH1
Trigger level = 2.5 V

```
' Turn CH1 to 3 measurement on and set the probe and range.
  for (i = 0; i < 3; i++)
  {
      ' Measurement = On
       Ret = sxAPI1.SetAcqSwitch(sxAPI1.ChHndl(hGrp, i), BOOL.ON);
      ' Probe = "1:1"
       Ret = sxAPI1.SetControl(sxAPI1.ChHndl(hGrp, i), ":CHAN<ch>:PROB 1");
      ' Range = 10 V
       Ret = sxAPI1.SetControl(sxAPI1.ChHndl(hGrp, i), ":CHAN<ch>:VDIV 1");
      ' (The value is set to 1/10 of the range because the range is
        specified using VDIV.)
       }

' Measuring mode = "Triggered"
  Ret = sxAPI1.SetAcqMode(hGrp, ACQMODE.TRIG);

' Trigger mode = "Single"
  Ret = sxAPI1.SetTrigMode(hGrp, TRIGMODE.SINGLE);

' Sample interval = 100 us (= 10 kS/s)
  Ret = sxAPI1.SetSamplingInterval(hGrp, 0.0001);

' Record length = 0.1 s
  Ret = sxAPI1.SetAcqSpan(hGrp, 0.1);

' Trigger position = 10%
  Ret = sxAPI1.SetTrigPos(hGrp, 10.0);
```

```
' Trigger source = CH1
  Ret = sxAPI1.SetControl(hGrp, ":TRIG:SIMP:SOUR CH1");

' Trigger level = 2.5 V
  Ret = sxAPI1.SetControl(hGrp, ":TRIG:SIMP:LEV 2.5");
```

Starting Measurement

The code below sends a command to the units to start measuring.

```
' Start measurement
  Ret = sxAPI1.AcqStart(hGrp);
```

Saving Waveform Data

The code below saves the acquired waveform data to a (WDF format) file.

```
' Save the most recent waveform data
  Ret = sxAPI1.SaveAcqData(sxAPI1.UnitHndl(hGrp, 0), -1, "C:\\Data\\
  filename");
'    Caution: -1 indicates the most recent data.
```

Closing

Closing and Disconnecting

The code below deletes the event handle, closes the unit group, and disconnects.

```
' Delete the event handle
  Ret = sxAPI1.DeleteEvent(hGrp);

' Close the unit group
  Ret = sxAPI1.CloseGroup(hGrp);

' Disconnect
  Ret = sxAPI1.Exit(hComm);
```

Index

A Page

acquisition data information 3-46
acquisition number, latest 3-46
asynchronous messages 1-3
auto naming 3-40
auto recording condition settings 3-3
auto recording controls 3-3
auto recording destination 3-33
auto recording on/off 3-26

B Page

binary data, send 3-20

C Page

calibration 3-54
channel handle 1-2, 3-14
channel label 3-31
channel number 3-15
commands, send 3-18
commands, send and receive 3-20, 3-21
comments 3-42
communication command controls 3-2
communication commands 4-1
communication execution errors 5-3
communication handle 1-2, 3-12
communication handle, retrieving 3-1
communication query errors 5-3
communication syntax errors 5-2
CPU temperature 3-54
current directory 3-61
current drive 3-60

D Page

debugging 3-5
development environment 1-1
device information retrieval 3-1

E Page

error code 3-59
event control 3-2
event handler 3-23
events 1-3, 2-1
exit 3-6

F Page

file, delete 3-63
file, divide 3-45
file information 3-63
file name 3-40
file order 3-41
file structure 1-1

H Page

handles 1-2
handles, closing 3-1
handles, opening 3-1

I Page

initialization 3-1, 3-6
instantaneous values 3-52
internal media operations 3-5

K Page

key lock 3-59

L Page

library errors 5-1

M Page

manual trigger 3-44
measured data, delete 3-4
measurement, start 3-43
measurement, stop 3-43
measurement condition settings 3-2
measurement controls 3-3
measurement on/off 3-25
measuring group handle 1-2, 3-13
measuring group number 3-16
measuring group settings 3-1
measuring mode 3-26
module handle 1-2, 3-14
module information 3-9
module number 3-15

O Page

operating environment 1-1

P Page

parameters, get received 3-21, 3-22
pre-trigger points 3-30
pre-trigger position 3-30

R Page

re-search 3-7
record count 3-39
recording, start 3-45
recording destination folder 3-39
recording interval 3-38
recording interval mode 3-37
recording interval points 3-38
recording start condition 3-33
recording start time 3-34
recording stop 3-45
recording stop condition 3-35
recording stop time 3-36
recording time 3-36
record points 3-37

S Page

sample points 3-28
sample rate 3-27
sampling clock 3-27
sampling time 3-29
self test, execute 3-54
setup data, initialize 3-53
setup data, load 3-53
setup data, save 3-53

setup data access..... 3-4
SNTP settings..... 3-58
syntax rules 4-1
system-related functions..... 3-4
system error..... 5-3

T

Page
trigger count..... 3-31
trigger mode 3-29

U

Page
unit, close 3-11
unit, open..... 3-11
unit errors 5-2
unit group, close 3-10
unit group, open..... 3-10
unit group handle..... 1-2, 3-12
unit group information..... 3-8
unit group number 3-15
unit handle..... 1-2, 3-12
unit information..... 3-9
unit number..... 3-15
units, number of..... 3-8

W

Page
waveform data, delete 3-52
waveform data, get..... 3-47
WDF file..... 3-50